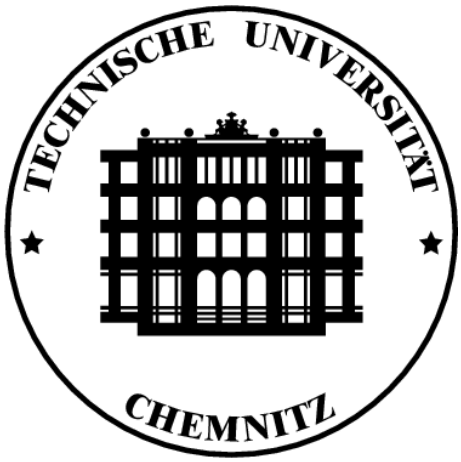
Chemnitz University of Technology



Faculty of Computer Science

Chair of Computer Architecture

Seminar Paper

MPICH-G2

René Grabner

rene.grabner@informatik.tu-chemnitz.de

30th September 2002

René Grabner **MPICH-G2**
Seminar Paper, Chemnitz University of Technology, 2002

Seminar paper: MPICH-G2

René Grabner

30th September 2002

Contents

1 Introduction

During the past years arising technologies and globalisation have forced institutions and companies dealing within similar fields to grow together. Effects of synergy should help in two kinds, first is to save money and second is to establish or expand influence on the world market.

In the field of computer technology a comparable process takes place. Computers became faster and got more storage, but the need for these resources has grown even much faster. A solution, which suggests itself, insists on coupling several non-expensive systems to form a more rapid one. The first successful undertaking known to public was the *Beowulf Project* [Beowu02] in 1994. A few years later many institutions and companies went away from expensive and 'proprietary' supercomputers to cluster systems built of rife standard components, so called commodity clusters. But during some development or research projects, there is often a temporary need for even more computational power and storage size. An acquisition of a new cluster system would be too expensive. An extension of the existing cluster system could in most cases be inefficient and/or cause a too heterogeneous environment. So why not coupling the existing cluster systems of two or more institutions for the time of elevated need? The idea to build clusters out of cluster systems was born.

But before, institutions want to define rules and policies about what is shared and who is under which conditions allowed to share local resources and, on the other side, to use lent, foreign resources. This leads to controlled and coordinated resource sharing and resource usage. The so called "Grid Problem" [Foste02a] is about issues that arise in such dynamic, scalable virtual organizations. Many research is done to develop environments, which try to provide possibilities to share distributed resources within a grid in a highly flexible way. Architectures, that have been developed, define protocols for basic mechanisms such as resource negotiation, establishing connections, managing and exploiting shared resources. By using open and established standards these environments aim to provide extensibility, portability and interoperability. At least the latter two are very important demands as grids are heterogeneous by nature. Hardware by different vendors, incompatible successors of some components that need to be replaced or extended, different software releases and various intra- and inter-connect solutions with inappropriate drivers for the current operating system are just a few examples one might face.

Skillicron [Skill01] summed up four types of Grids: computational grids, access grids, data grids and data-centric grids, each with a different designation. But, when speaking about grids, as in all of these cases, one has in mind systems, which generally can be distributed world-wide. For instance, they involve government laboratories and large national facilities and authorities. Beyond this, in practice three more types of coupled, distributed systems can be determined: national grids, institutional or campus grids and cluster or local grids. The term 'grid' is in both latter cases inappropriate, it is better to use the more precise term it cluster-of-clusters system. Foster and Kesselman [Foste99] show up parallels between (computational) grids and the electric power grid, which "...provides power to billions of devices in a relatively efficient, low-cost, and reliable fashion." Both of them span wide

areas. But what are the key aspects of a grid? Foster answers this question in his three point checklist [Foste02b].

- A grid system manages resources that are not maintained by centralized control. Resources integrated by a grid reside within several administrative domains.
- A grid system uses of standard, open, general-purpose protocols and interfaces. Issues addressed by these protocols and interfaces are such as authentication, authorization, resource discovery and resource access.
- A grid system delivers nontrivial qualities of service to its users. Examples are response time, throughput, availability and security. Further on the co-allocation of multiple resource types to provide the basics for complex user demands, which aim to achieve the benefit of the grid system to be significantly greater than the benefit of the sum of its parts.

Grids and cluster-of-clusters systems have a lot of things in common. But, the main differences occur because of locality. As cluster-of-clusters systems are set up within one institution or even in the same department, the demands on security and authentication might be different or less important, because of a secure environmental infrastructure. Generally, local or campus systems are of the computational type. Hence, these systems have interconnect networks with high bandwidths and relatively low latencies. grid environments and programming models, which are being developed for world-wide grids, might have performance losses in campus or local cluster-of-clusters systems. Their most important factors lay on overcoming heterogeneity and securely inter-connecting distributed resources.

There are several models for metacomputing: the parallel object-oriented approach, special shared memory and message passing. The latter is still important and focused in this study. Meta-computing environments introduce new challenges not known in sequential or parallel programming. Multiple administrative domains and new failure modes make it a new challenge to debug meta-computing programs. As well do large and often unpredictable variations in performance of machines and inter-connect networks. But, in spite of these, the basic programming problem is not fundamentally different.

Here is a short overview of the structure and the content of the seminar paper.
This study examines in first place the performance of MPI-operations, that are implemented by MPICH-G2 on top of the Globus Toolkit 2.0. Further on the usability of the Globus Toolkit in non-world-wide grids such as campus or local cluster-of-clusters environments is determined. The following sections start with describing the basic functionality and priciples of the software that will be used. It follows a section that documents and describes the installation procedures for these software packages. Hereafter several general issues of network performance measurements in distributed systems are discussed before the benchmarking suite that will be used following. Then the most interesting results found during the tests are presented, separated into several test cases. Summaries and short conclusions finalize each of these sections. After all of that one section explains problems that have been found and experiences that were gained. Finally, the appendices contain material, that did not fit into the text, but still is important to the wholeness of this work.

2 Software Overview

2.1 Globus Toolkit

The Globus Toolkit is an open source software toolkit with an open architecture. It is being developed mainly by the Mathematics and Computer Science Division at Argonne National Laboratory and contributed by lots of developers world-wide. Many companies support the open source activities, for instance *IBM* shortly announced an enhanced version of the Globus Toolkit with special support for eServers running Linux or AIX. [IBMan02].

The Globus Toolkit is a collection of software tools that provide basic services for building computational grids and appropriate grid-based applications. The toolkit is based on three main components, often refered to as pillars. Figure 1 illustrates the components.

- *The Resource Management Pillar*
  The first pillar of the Globus Toolkit provides Resource Management. The Resource Management involves the allocation and management of grid resources. It includes components like GRAM, DUROC and GASS.
- *The Information Services Pillar*
  The second pillar of the Globus Toolkit is Information Service, which provides information about grid resources. This area includes MDS, which in turn provides the GIIS and GRIS components.
- *The Data Management Pillar*
  The third pillar of the Globus Toolkit is Data Management. The Data Management involves the ability to access and manage data in a grid environment. This includes components such as GridFTP, which is used to move files between grid-enabled storage systems.

All of these components use services provided by the GSI security protocol at the connection layer. Since version 2 one can download and install each component separately in a client and a server version. Support for that is given by the grid Packaging Toolkit that is developed

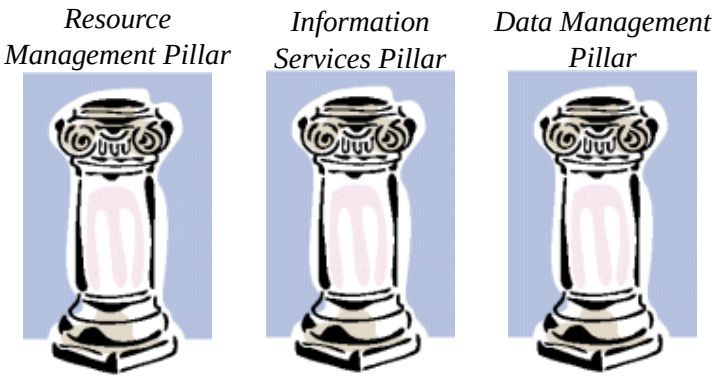


Figure 1: Globus Toolkit pillars

in collaboration with the National Center for Supercomputing Applications (NCSA) at the University of Illinois.

Within the following short parts an overview of the components mentioned above is given.

Grid Security Infrastructure (GSI)
As the Globus Toolkit can connect different sites over open networks to grid systems, there is the need for mechanisms that enable secure authentication and communication. GSI provides several useful services for grids, including mutual authentication and single sign-on. The bases for GSI are public key encryption, X.509 certificates and the Secure Socket Layer (SSL) communication protocol. Some enhancements to these standards were necessary to support single sign-on and delegation. The GSI implementation of the Globus Toolkit conforms to the Generic Security Service API (GSS-API). This is a standardized API for security systems from the Internet Engineering Task Force (IETF).

Resource Management
The Resource Management Architecture of the Globus Toolkit is a layered system. It contains high-level global resource management services on top of local resource allocation services. In figure 2, that shows an overview of the components in this architecture, one can see three main components:

- An extensible Resource Specification Language (RSL) [GTeam02d], which provides a method for exchanging information about resource requirements between all of the components in the Globus resource management architecture. It is used to describe jobs as shown for instance in section 3.1.3 on page 13.
- An interface to all of the various local resource management tools like LSF, NQE, LoadLeveler and Condor. That interface is provided by the Globus Resource Allocation Manager (GRAM).
- The co-allocation service is named Dynamically-Updated Request Online Coallocator (DUROC). It coordinates single requests that can span multiple GRAMs.

Information Management
The information management pillar is implemented by the Globus Metacomputing Directory

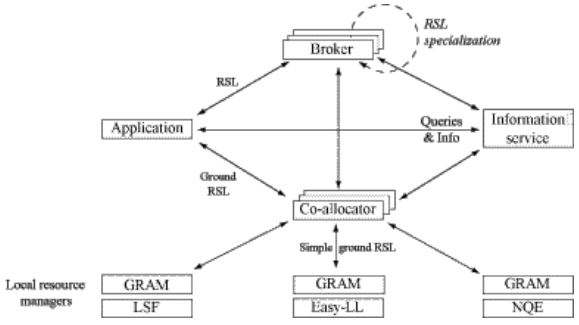


Figure 2: Globus Toolkit resource management architecture

Service (MDS). It provides the necessary tools to build an LDAP-based information infrastructure for computational grids. MDS uses the LDAP protocol as basis for queries about system information of a large number of system components. It also can construct unique resource namespaces for system resource information that are distributed over, for instance, several organizations.

Data Management
A secure, high-performance and reliable data transfer protocol, that is optimized for high-bandwidth wide-area networks is the center of the Data Management pillar. It is called GridFTP and is based on FTP, but provides enhanced features required by data and also computation grid projects. Among others, these are GSI security on control and data channels, the possibility to bundle channels for parallel transfers, partial file transfers and direct server-to-server transfers.

Much more information than this short overview can be found on several Globus-related web sites, beginning from Globus' homepage at [GTeam02a].

2.2 MPICH

There is no doubt that MPICH is the most important MPI implementation, which is freely available and portable. It is being developed by Argonne National Laboratory and Mississippi State University. The first version of MPICH had been developed in parallel to the first version of the Message Passing Interface standard. It was a kind of proove that the standard does not becomes too complex and can be implemented at once. In May 1994 both, the final version 1.0 of the MPI standard and the first version of the MPICH implementation, were complete. And MPICH was fast and portable.
The portability of MPICH originates from its design. There are two layers. The major part of the code is implemented device independently on top of the Abstract Device Interface (ADI). This makes it possible to easily port MPICH to new hardware architectures. The minor, device dependent part implements the ADI. It should preferably be implemented by the hardware vendor for a maximum of efficiency. Each such implementation is called an *MPICH device*. The current version number of ADI is 2, but there is already a draft for the next version ADI-3 with many enhancements. It aims to provide services for the support of the MPI-2 standard. Conventional networks, as previously, and additionally networks with remote memory access such as *Infiniband* shall be supported.
Many vendors such as Digital, Intel, Sun, SGI and Fujitsu took advantage of the portability of MPICH brought by the ADI. They developed special implementations for their own machines and platforms. The result is an increasing number of very efficient MPI implementations, which increase the propagation of MPI and MPICH.

2.3 Globus-enabled MPI: MPICH-G

As mentioned in section 2.1 on page 3 the Globus Toolkit enables access to various computational resources, but it cannot itself provide a convenient way to use several resources simultaneously. A Globus-enabled version of MPICH can solve this by providing a *globus*-device. The first version, called MPICH-G, is based on the Nexus Communication Library.

The Nexus library was the basis for all communications in MPICH-G. It supports multiple protocols including a highly efficient TCP implementation and automatic datatype conversions between different architectures. But, very all message passing is done by the Nexus library, which leads to performance losses for instance in intra-machine communication. Data from the sending application's source buffer needs to be copied to the Nexus buffer before sending. On receiving data from the Nexus buffer need to be copied to the application's destination buffer. So there was the need for improvement.

2.4 Globus-enabled MPI: MPICH-G2

The current implementation - the *globus2*-device - makes no longer use of the Nexus library. It is said to have re-implemented the 'good' parts of Nexus and improved the others [Allan01]. MPICH-G2 is supported by the Globus Toolkit since version 1.1.4 and conforms to the MPI standard 1.1 with some additional features of MPI 2.0.

In the example above, the additional copies of the buffers from the application to the communication layer and vice versa are removed, data flows directly between the applications' buffers, which speeds up MPI performance by decreasing latency.

There are several more new features to MPICH-G2, for instance the ability to specify the IP addresses or network interfaces to be used, to specify IP port ranges if there are firewall issues and to tune the TCP buffer sizes. All of these can be done in resource specification language while describing the jobs and their requirements. Two more new features will be explained in short on the following lines.

Topology aware operations In cluster-of-clusters and grid environments computational resources are connected by short distance and fast local area networks and by longer and slower wide area networks. This results in very different communication speeds between resources. The idea is, to separate communication methods into groups, each with similar properties. MPICH-G2 defines the following communication protocols, which order the various communication methods based on their performance (from slowest to fastest):

WAN-TCP < LAN-TCP < intra-machine TCP < vendor-supplied MPI

In turn MPICH-G2 is able to maximize communication over the fastest links and to minimize the communication over the slowest links. The current version of MPICH-G2 uses this approach to implement some collective MPI operations that are topology-aware, for instance `MPI_Barrier` and `MPI_Bcast`. Usage and improved measurement results due to topology-awareness are presented in section 5.4 on page 33.

Topology Discovery Mechanism MPICH-G2 not only uses information about the topology of the underlaying communication methods for its own purposes, but it also provides these information to the application programmer. So one is able to dynamically adopt an application to the current network infrastructure by asking MPICH-G2 for this information. For instance,

one can create MPI communicators that only contain processes, which can communicate via the same communication method. These topology information are provided by two vectors, each with a length of the current MPI communicator size: *MPICHX_TOPOLOGY_DEPTHS* and *MPICHX_TOPOLOGY_COLORS*. So an MPI application can find out which processes (which ranks) can communicate at the same level. An example and further descriptions can be found on the MPICH-G2 homepage [MG2Te02].

An overview of performance comparisons between MPICH-G, MPICH-G2 and vendor supplied implementations can be found on the MPICH-G2 homepage [MG2Te02].
Later in this study, one will see that there is still work to do and enough space for improvements on MPICH-G2. In section 5.2 on page 24 MPICH-G2 is benchmarked on a multiprocessor machine and compared to the MPICH-device *ch_shmem*.

3 Software Setup

This section deals with basic software issues needed to perform performance measurements of MPICH-G2. The first part covers the complete set up of the Globus2 Toolkit. Based on these results the second part is about setting up MPICH-G2 on top of the Globus2 installation.

3.1 Globus2: Installation, Configuration, Customization

Globus2 is the first release of the Globus Toolkit which uses NCSA's Grid Packaging Technology (GPT). As mentioned earlier, the Globus Toolkit consists of three main parts: Resource Management, Information Services and Data Management. According to this division there are three pillars available for download, each as a binary and a source bundle. When one intends to build applications on top of Globus2, it is neccessary to choose the appropriate ones out of the source bundles. However, this study focuses on performance measurements of MPICH-G2. So Client-, Server- and SDK-bundle from the Resource Management pillar will do.

For installing and maintaining Globus2's packages the Grid Packaging Toolkit must be installed first. It is also available on Globus2 download page. The following sections provide some more information for the procedure of installation and configuration than given in [GTeam02c], [GTeam02b]. Especially customizations to local conditions will be described. This section is divided into three parts. The first is meant for the administrator, it explains his tasks while installing. Within the next shorter part the customizations to be done by the user will be explained. Finally, there is a part which performs some basic test to verify the installation.

3.1.1 Administrator's Part

When finished downloading all required files the installation is straight forward and could be done by a script similar to the following. It is recommended to have a separate user, for

instance called *globus*, for installation.

```
#!/bin/bash

export GLOBUS_LOCATION="/ra/projects/packages/globus-2.0/globus"
SRC_DIR="/ra/projects/packages/globus-2.0/src-tgz/globus2"

mkdir -p $GLOBUS_LOCATION

tar -xzf $SRC_DIR/dirgpt-1.0.tar.gz
cd gpt-1.0

#install Grid Packaging Toolkit
./build_gpt

#install Globus2 packages (resource-management from source)
for PCKG in $SRC_DIR/globus_resource_management_bundle-*-src.tar.gz
do
    $GLOBUS_LOCATION/sbin/globus-build -install-only \
                                       -log=./build.log \
                                       $PCKG "gcc32dbg"
done

#run some postinstall-scripts
$GLOBUS_LOCATION/sbin/gpt_verify
$GLOBUS_LOCATION/sbin/gpt-postinstall
```

First, one has to figure out where to install the applications. In the case that several machines of one cluster will be integrated into the Globus2 services, it is recommended to choose a destination on a file-server, which can be accessed by all of the other nodes. According to this destination directory the most variable `GLOBUS_LOCATION` has to be set. The creation of this destination directory is the next step before extracting and installing GPT, which in turn installs the desired Globus2 packages. The last argument is the desired flavor of Globus2. Since performance measurements will be done one is recommended not to choose a debug flavor (as *gcc32dbg*). Running the verification script ensures the success of the previous steps, the postinstall script does some automatic customizations.

During the next step the Grid Security Infrastructure has to be configured by running the GSI-setup script. When installing for a couple of nodes within a private cluster as in this case, it is advisable to keep security related files within the central repository of the file-server as well. So one runs the script as user *globus* with the parameter `non-root`.

```
$GLOBUS_LOCATION/setup/globus/setup-gsi -nonroot
```

When asked for base distinguished names one has to enter the appropriate information. In this case there already exists a certification authority, and change the defaults to be read as follows.

```
(1) Base DN for user certificates
        [ c=de, o=tu-chemnitz ]
(2) Base DN for host certificates
        [ c=de, o=tu-chemnitz ]
```

Due to a bug the initial gatekeeper configuration file contains some wrong pathnames, which have to be fixed. So the next step is to edit this file named `$GLOBUS_LOCATION/etc/globus-gatekeeper.conf` to be read as follows.

```
-x509_cert_dir /ra/projects/packages/globus-2.0/globus/share/certificates
-x509_user_cert /ra/projects/packages/globus-2.0/globus/etc/grid-security/hostcert.pem
-x509_user_key /ra/projects/packages/globus-2.0/globus/etc/grid-security/hostkey.pem
-gridmap /ra/projects/packages/globus-2.0/globus/etc/grid-mapfile
-home /ra/projects/packages/globus-2.0/globus
-e libexec
-logfile var/globus-gatekeeper.log
-port 2119
-grid_services etc/grid-services
-inetd
```

Subsequent to creating the directory `$GLOBUS_LOCATION/etc/grid-security` the host certificate can be set up. Again, in this case there are already signed host certificates from previous versions of the Globus Toolkit for some machines which can be used furthermore. To accomodate the certificates within one directory shared among all nodes the file-server used is the Cluster-NFS server [Warne02]. The files renamed might look similar to these:

```
[globus@john grid-security]$ ls -1
hostcert.pem$$IP=192.168.1.10$$
hostcert.pem$$IP=192.168.1.11$$
hostcert.pem$$IP=192.168.1.12$$
hostcert.pem$$IP=192.168.1.9$$
hostkey.pem$$IP=192.168.1.10$$
hostkey.pem$$IP=192.168.1.11$$
hostkey.pem$$IP=192.168.1.12$$
hostkey.pem$$IP=192.168.1.9$$
```

If one wants to add more nodes, more certificates would be necessary. The certificate request must be generated as user *root* for these machines. This can be done by running the following command with the appropriate hostnames.

```
grid-cert-request -host jack13.oscar \
                        -key $GLOBUS_LOCATION/etc/grid-security/jack13_hostkey.pem \
                        -cert $GLOBUS_LOCATION/etc/grid-security/jack13_hostcert.pem \
                        -req $GLOBUS_LOCATION/etc/grid-security/jack13_host.req
```

The certficate request needs to be signed by the certification authority (see below). Finally, one can rename the hostkey files, which contain the private key for the host and the host certificate file in the terms of Cluster-NFS' naming conditions to assign to the specific machine.

One has to make sure, that the files containing the private host keys must not be readable by anyone else than root. If the NFS-Server squashes user-id's of 0 to user nobody, the shared directory has to be exported using the `no_root_squash` option in `/etc/exports`.

To make Globus2 accept the host- and later on the user-certificates the signing Certification Authority (CA) has to be announced. This can be achieved by copying the CA's certificate to `$GLOBUS_LOCATION/share/certificates`. The file's name is the hash-value of the certificate. Additionally a signing policy, which indentifies the CA, has to be set up. This policy gets appended to the file `ca-signing-policy.conf` within the certificate directory. In this local case the CA's name is *RA-CA*.

```
[globus@john certificates]$ ls -1
ca-signing-policy.conf
e64c5fc0.0
e64c5fc0.signing_policy

[globus@john certificates]$ cat ca-signing-policy.conf
# EACL entry #1  - RA-CA
 access_id_CA    X509     '/C=de/O=tu-chemnitz/CN=RA-CA'
 pos_rights      globus   CA:sign
 cond_subjects   globus   '"/C=de/O=tu-chemnitz/*" "/C=DE/O=TU-Chemnitz/*" "/C=de/O=Grid/*"'
```

One more change has to be made to introduce the local CA. In the file `$GLOBUS_LOCATION/etc/grid-security.conf` the name of the CA and its e-mail address are to be adjusted. The section might read like:

```
SETUP_GSI_HOST_BASE_DN="c=de, o=tu-chemnitz.de"
SETUP_GSI_USER_BASE_DN="c=de, o=tu-chemnitz.de"
SETUP_GSI_CA_NAME="RA-CA"
SETUP_GSI_CA_EMAIL_ADDR="daniel.balkanski@informatik.tu-chemnitz.de"
```

When a job gets started on a foreign node within a grid, Globus needs a mapping from worldwide unique user- and proxy-certificates to users, who are local to the node, which finally executes the job. This is done by the file `$GLOBUS_LOCATION/etc/grid-mapfile`. It might have several lines similar to this:

```
"/C=de/O=tu-chemnitz/CN=Rene Grabner" regra
```

In order to execute jobs later on each node must provide the gatekeeper service. In previous versions of the Globus Toolkit the gatekeeper was normally a standalone daemon started at system boot-up. Now the gatekeeper is started by `inetd` or preferably by `xinetd`. So there is some effort to be done for each node of the cluster. Depending on the installation it might be neccesary to log in on each node and perform the modifications locally or, when having the machines booting diskless from a file-server, the modifications can be done from one machine for all nodes. Another approach could be the usage of `cfengine` [Burge02]. But this is beyond the scope of this study.
On Red-Hat Linux based systems one has to copy a file similar to the following into the directory `/etc/xinetd.d`.

```
service globus-gatekeeper
   {
      socket_type = stream
      protocol    = tcp
      wait        = no
      user        = root
      server      = /ra/projects/packages/globus-2.0/globus/sbin/globus-gatekeeper
      server_args = -conf /ra/projects/packages/globus-2.0/globus/etc/globus-gatekeeper.conf
      env         = LD_LIBRARY_PATH=/ra/projects/packages/globus-2.0/globus/lib
      disable     = no
   }
```

When Globus2 binaries were built using dynamic libraries, which is the default, it is necessary to have the environment variable `LD_LIBRARY_PATH` set pointing to the directory containing the shared libraries. This is done by the xinetd-process before starting the gatekeeper upon an incoming request.

Now, add the service-name to port translation entry to the file`/etc/services`, which might look like this:

```
globus-gatekeeper      2119/tcp                  # Globus Gatekeeper
```

Afterwards make the xinetd-process reread its configuration either by restart or sending the appropriate signal.

Because of working with a shared installation using Cluster-NFS, one has to create the log-files, which will be used by the gatekeeper-service. This might look like:

```
[globus@john var]$ touch globus-gatekeeper.log\$\$IP\=192.168.1.9\$\$
[globus@john var]$ touch globus-gatekeeper.log\$\$IP\=192.168.1.10\$\$
[globus@john var]$ touch globus-gatekeeper.log\$\$IP\=192.168.1.11\$\$
[globus@john var]$ touch globus-gatekeeper.log\$\$IP\=192.168.1.12\$\$
```

Do not forget the escaping backslash when for instance using a Bourne Shell.
The last step is to run the postinstall script `$GLOBUS_LOCATION/sbin/gpt-postinstall` once again. Now that all issues concerning security are configured, the script tries to verify the installation and finally creates an entry for the jobmanager within the jobmanager file (`$GLOBUS_LOCATION/etc/grid-services/jobmanager`). Standard is the simple fork-jobmanager. As its name reveals the gatekeeper will fork itself on each node to start the job's process. Alternatives are kinds of batch systems as Portable Batch System (PBS) [Verid02] and especially for High Throuput Computing Condor [Grabn02], [CTeam02] and its grid-enabled variation Condor-G [Frey01].

At this point, the administrator's tasks for installing the Globus2 Toolkit are finished.

3.1.2 User's Part

Before one can start working with Globus2 under one's user-account two things have to be done. At first there must be set some environment variables to point among others to Globus specific directories. This can be done easily by modifying the login-shell's profile. In this case when using a kind of Bourne Shell the file `.bash_login` might look similar to this:

```
export GLOBUS_LOCATION="/ra/projects/packages/globus-2.0/globus"
source $GLOBUS_LOCATION/etc/globus-user-env.sh
export PATH="$GLOBUS_LOCATION/bin:$PATH"
```

The second step is to set up a certificate. When already having a certificate from previous Globus installations one has to verify that the certificate and private key file is located within one's home-directory in `~/.globus/`. Make sure the private key is readable by the user only. To request a new certificate one can use the command `grid-cert-request` and send the cert-request as described in [GTeam02b] to the Globus-CA or preferably a local CA.

3.1.3 Verification and first Tests

Finally, one can run a test to see if Globus2 is working on all machines properly. To do this log in as user and generate a proxy-certificate, which will be used to authenticate against all nodes. Simply run the following command:

```
bash-2.05$ grid-proxy-init
Your identity: /C=de/O=tu-chemnitz/CN=Rene Grabner
Enter GRID pass phrase for this identity: ******
Creating proxy ............................................ Done
Your proxy is valid until Thu Jun 27 02:10:13 2002
```

As there is no application developed in this study so far the GNU `date` utility will be used first and given to `globusrun` as application to be executed.

```
bash-2.05$ globusrun -o -r jack9 '&(executable=/bin/date)'
Wed Jun 26 15:04:41 CEST 2002
```

The second test is a self-compiled and extended Globus version of Kernighan and Ritchie's famous "hello, world" program. The function call `globus_duroc_runtime_barrier()` returns, when all job processes have successfully started. The rest of the source code is self-explaining. Detailed information on programming on top of Globus2 can be found within the Globus Toolkit API Reference [GTeam02e].

```
#include <globus_duroc_runtime.h>
int main(int argc, char *argv[])
{
    int rankp, sizep, rc;
    char hn[256];

    globus_module_activate(GLOBUS_DUROC_RUNTIME_MODULE);
    globus_duroc_runtime_barrier();

    globus_libc_gethostname(hn,256);
    globus_assert(rc == GLOBUS_SUCCESS);

    globus_duroc_runtime_intra_subjob_rank(&rankp);
    globus_duroc_runtime_intra_subjob_size(&sizep);

    globus_module_deactivate(GLOBUS_DUROC_RUNTIME_MODULE);
    printf("hello, world. I am subjob %i out of %i on node %s\n",rankp,sizep,hn);
}
```

This small example comes along with a `Makefile`. Its main purpose is to include a system specific header:

```
include makefile_header

hello:
        $(GLOBUS_CC) $(GLOBUS_CFLAGS) $(GLOBUS_INCLUDES) -c hello.c
        $(GLOBUS_LD) -o hello hello.o \
        $(GLOBUS_LDFLAGS)  \
        $(GLOBUS_PKG_LIBS) \
        $(GLOBUS_LIBS)

clean:
        /bin/rm -rf *.o hello
```

Using the Globus2 flavor as specified during Globus2 installation 'gcc32', the required header file can be generated by executing the command:

```
$GLOBUS_LOCATION/sbin/globus-makefile-header -flavor=gcc32          \
  globus_common globus_gram_client globus_io globus_data_conversion \
  globus_duroc_runtime globus_duroc_bootstrap > makefile_header
```

Next, the binary can be build by `make hello`. To execute the application as a Globus2 job a file describing the job in the Resource Specification Language is needed. In the following example the application gets executed twice `(count=2)` on both nodes *jack9* and *jack10*.

```
+
( &(resourceManagerContact="jack9")
   (count=2)
   (label="subjob 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
   (LD_LIBRARY_PATH /ra/projects/packages/globus-2.0/globus/lib/))
   (directory=/home/ra/regra/globus_hello-world)
   (executable=/home/ra/regra/globus_hello-world/hello)
)
( &(resourceManagerContact="jack10")
   (count=2)
   (label="subjob 2")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
   (LD_LIBRARY_PATH /ra/projects/packages/globus-2.0/globus/lib/))
   (directory=/home/ra/regra/globus_hello-world)
   (executable=/home/ra/regra/globus_hello-world/hello)
)
```

Finally, one can start the jobs and see the following output.

```
bash-2.05$ globusrun -f hello.rsl -o
hello, world. I am subjob 1 out of 2 on node jack9.oscar
hello, world. I am subjob 0 out of 2 on node jack9.oscar
hello, world. I am subjob 1 out of 2 on node jack10.oscar
hello, world. I am subjob 0 out of 2 on node jack10.oscar
```

Both tests explained above could be performed involving all nodes running a Globus2 gatekeeper to make sure the installations work properly.

3.2 MPICH-G2: Installation

Installation of the grid enabled MPI implementation MPICH-G2 is shorter and easier than the torture described above. The structure of the section is similar to the previous one.

3.2.1 Administrator's Part

As mentioned earlier MPICH-G2 is based on the MPI implementation MPICH from Argonne National Laboratory [Mpich02]. So the first step is to download a recent version of MPICH, it must be at least version 1.2.3 to work together with Globus2. After extracting the archive containing the source files the configuration has to be done by running `./configure`. Beyond the (optional) path prefix of the installation directory one has to specify the MPICH-device as *globus2* and the flavor of Globus2 used while installing the toolkit, refer to section 3.1.1 on page 8. The command to be executed might be similar to this:

```
./configure --prefix=/ra/projects/packages/globus-2.0/mpich-g2 \
            --with-device=globus2:-flavor=gcc32
```

When MPICH was successfully configured the next steps are `make` and afterwards `make install`.
Notice: MPICH did not compile successfully when running `make` with the argument `-j 2` to start two concurrent jobs on the dual-processor machine.

3.2.2 User's Part

The work to be done as user is also very few. Comparable to Globus2 above the environment variable `PATH` is to be extended to the path of the MPICH-G2 binaries, which includes among others the compiler `mpicc` and `mpirun`. To extend the path one adds a line similar to the following to its profile or, as in this case, to `.bash_login`.

```
export PATH="/ra/projects/packages/globus-2.0/mpich-g2/bin:$PATH"
```

3.2.3 Verification and first Test

The following example is a MPI version of the test presented in section 3.1.3 on page 12.

```
#include <mpi.h>
int main(int argc, char *argv[])
{
    int rankp, sizep;
    char hn[256];

    MPI_Init(&argc,&argv);

    gethostname(hn,256);

    MPI_Comm_rank(MPI_COMM_WORLD,&rankp);
    MPI_Comm_size(MPI_COMM_WORLD,&sizep);

    printf("hello, world. I am subjob %d out of %d on node %s\n", rankp,sizep,hn);
    MPI_Finalize();
}
```

The compilation is simple, just run MPICH-G2's `mpicc` with the source file as argument, for instance:

```
mpicc hello.c -o hello
```

Similar to MPICH, applications compiled for MPICH-G2 can be started by using `mpirun` - the MPICH-G2 version. There has to be a file `machines` describing how many processes of the application should be created on which nodes (normally the number of processors installed). This file might look as follows:

```
"jack9.oscar" 2
"jack10.oscar" 2
```

Afterwards one can use `mpirun -np 4 hello` to start four processes on two machines using all of the four processors.
As MPICH-G2 is set up on top of Globus2, job starts have to be passed to the job starting facilities of the Globus Toolkit. One can use `mpirun` with the additional `-dumprsl` option. This causes a job description in RSL to be written to standard out, which could preferably be redirected to `hello.rsl`. Now, if desired, one can view and modify the RSL script and afterwards use `mpirun -globusrsl hello.rsl` to start. Alternatively `globusrun -f hello.rsl -o` can be used, which in turn, would be executed by `mpirun`.
The output of the small test application shows the semantic difference between this one and the Globus example on page 13. Here all participating processes are bundled within one communicator: *MPI_COMM_WORLD*.

```
bash-2.05$ mpirun -globusrsl hello.rsl
hello, world. I am subjob 1 out of 4 on node jack9.oscar
hello, world. I am subjob 0 out of 4 on node jack9.oscar
hello, world. I am subjob 3 out of 4 on node jack10.oscar
hello, world. I am subjob 2 out of 4 on node jack10.oscar
```

4 Performance Measurement

The first part of this section is about performance measurements in general. It lists some classes of problems and common mistakes when trying to measure performance in distributed systems and gives hints how to avoid them. The second part is about approaches of measurement. It distinguishes benchmarking from profiling and introduces several tools for each of them.

4.1 Classes of Problems

There are several classes of problems and common mistakes, which occur when performing communication performance measurements in special or time measurements in general. These classes are issues with the test design, with the tested functions and with the final interpretation of the data collected during the tests. See also [Gropp99], [Gropp02].

4.1.1 Problems with Test Design

- *Establishing of initial communication links*
  Starting up applications and establishing communication links take up a lot of time in many environments. This might be due to the communication hardware and the network or due to the software and protocols used. When performing communication time measurements one has to be sensible of the time it takes to establish connections and the number of them. For instance some systems dynamically establish and close connections. A method that can be used, if possible, is synchronizing all the processes involved and starting time measurements afterwards.

- *Influence of unrelated applications or jobs*
  Foreign traffic caused by unrelated file transfers or other data-intensive jobs on the system area network or wide area network, which is the intra- or inter-cluster communication network coupling clusters causes in most cases falsification of the performance measurement results.

- *Clock issues*
  There are several issues to the clock that is used to measure the elapsed time between two or more events. For instance: The frequency of event occurrence must be low relative to the resolution of the clock. If it is not, timing single events would lead to imprecise or even false results. Preferably a cycle counter, if present, could be used for timing. In general they provide the most fine grained resolution as the counter is incremented each clock cycle of the processor. But this leads to another problem in distributed systems. The clock cycle counter is local to one system. It would be good to

be able to measure the time between the sending event of a message on one system and the receiving event of this meassage on another machine. The problem is the absence of a global high resolution clock. Only high-end muli-processor systems with vendor supplied system area networks intraconnecting the subsystems, provide such a global high resolution clock.
One more issue is to try to substract an estimated duration of the function call that reads the clock from the result measured. This estmated duration is calculated by taking the average time it takes to do numerous such function calls. This leads to an artificially improved performance result.

➟ *Measuring with just two processes*
This may lead the results to be too optimistic as several systems poll on the number of possible message sources. Within a real configuration the performance might be much poorer. For instance, within an environment that uses an older version of Globus with a flavor of vendor-supplied MPI and MPICH-G on top of this, the Nexus library polls for incoming data on all of the possible sources: vendor-supplied MPI and TCP in a roundrobin fashion. As TCP needs long timeouts this slows down communication taking place just over vendor-supplied MPI.

4.1.2 Issues of the tested Functions

➟ *Ignore non-blocking calls*
Many environments also provide beyond the blocking function calls also non-blocking pendants. The usage of non-blocking calls to communication functions might lead to improved performance. They provide the possibility of several overlaping communications and of overlaping communication with proceeding calculation. This entails for instance latency hiding. Often the system can improve performance if it dynamically schedules communication.

➟ *Disregard effects of caches*
Various caches can influence the communication performance measured with benchmarking tools. They often send data coming from buffers residing in memory. A benchmarking tool always read and writes from/to the same buffer. If, for instance, the addresses of consecutive read operations are equal the data will come from the processors cache instead of the main memory.
Another issue is the following. When is a receive operation seen as completed? When the meassage has been completely received and the application notified, when the data has just been copied to memory or when the message data got stored in a cache. Each of these scenarios might lead to very different timing results and it is often hard to tell which scenario is actually present.

4.1.3 Comparing Apples and Oranges

➟ *Compare different approaches*
Do an apples to oranges comparison, for instance by comparing message-passing to shared memory copy. Both approaches are used for data transfers, but message passing

includes synchronization to indicate the availability of data while this step is often neglected in shared memory when comparing.

- *Compare CPU time to elapsed time*
  CPU time is the effective time a process has been active on a processor. Elapsed time is what one usually understands as elapsed time, which can be measured by an external clock. When a process for instance waits for data to arrive, on most systems it will fall asleep and get woken up as data has been received by the system. While the program is sleeping CPU time does not increase. So measuring the time for delivering a message, only elapsed time leads to correct results. Though CPU time is useful additional information, too.

- *Confusion of bandwidth*
  When measuring communication bandwidths one has to distinguish between the total or bisection bandwidth and the point-to-point bandwidth between to nodes. The ideal case within a dedicated switched network would be $bisectionbw = \frac{p2pbw \cdot numberofnodes}{2}$. Further more it is inopportune to compare switched and/or dedicated networks with shared network fabrics.

- *Communication pattern*
  Performing measurements with another communication pattern than the one used by the applications to be run on the system can be misleading. For instance, many scientific applications send messages head-to-head, so measuring ping-pong messages might lead to less useful and missleading results. An influence on performance can also be due to numerous messages between different processes.

- *Neglect of variations*
  In practice the minimum duration of an operation is often less important than the average case and the maximum. For instance, there are systems that might have good minimum timings, acceptable timings for the average case, but unacceptable variation with high peaks. Especially this is important to know, if (soft) realtime applications are to be run on that particular system.

4.2 Measurement Approaches

There are two different approches to measure performance: benchmarking and profiling. Benchmarking is done by using a special tool or toolkit, which performs numerous iterations of computation and/or communication and measures the elapsed time. They are used to measure the perfomance of levels underneath the application level, for instance the performance of the MPI-implementation, the performance of the network drivers and devices. Most interesting results are latencies and bandwidths and their course for various packet or messages sizes.
Profiling addresses mainly the level of the application itself. It is used to measure the performance of a given application that mostly needs to be relinked against the profiling libraries. This adds time measuring and logging code. After the application has been rerun, one can retrace and reproduce how much time had been spent in which function call. This is useful for both application developer and communication layer- / middleware developer to find

bottlenecks and inefficient implemented functions. A disadvantage of profiling tools is, that the profiling library cannot make sure to not influence the profiled process when flushing log data to disk as concurrent communication might take place.
The following episodes give some examples for both approaches.

4.2.1 Benchmarking Tools

Perftest The *perftest*-package is provided along with the MPI-implementation MPICH by the Mathematics and Computer Science Division at Argonne National Laboratory. The current version, used in this study, is 1.2. The package contains a few tools to measure the performance of a message passing environment. Although it comes with MPICH it can be used in combination with any MPI-implementation. The two major programs are `mpptest` for measuring point-to-point communication and `goptest` for measuring collective communication. Both provide a number of options to run a wide variety of tests. The most important of them, which will be used in this study, will be explained later in the section of benchmarking 5.1. One of the most outstanding point is that *perftest*-tools aim to yield reproducible results. Each test is performed several times to avoid misleading results as for instance increased initial latencies due to link establishment. Another feature is a function that adoptively calculates message sizes as the performance of data copies is not just a function of meassage length, but it often consists of several abrupt steps when designated message sizes are crossed. For instance many MPI-implementations switch over to a different protocol as the meassage size is higher than a fixed threshold. Hence, the `mpptest` program tries to generate a graph output that approximates the behaviour of the system best.

SKaMPI The *SKaMPI*-package (Special Karlsruher MPI-) benchmark has been developed at University of Karlsruhe, Germany. Currently the third realease of the package is available. The benchmark is very flexible and can be used to test various communication patterns, point-to-point and collective ones. It also can automatically specify message sizes while testing is in progress. A more practise oriented test than `mpptest`'s bisection test is the measurement with the *Master Worker Scheme*. One of the processes, the master, dispatches concurrent subtasks to all other processes (the workers), which send a reply for each message received from the master. The result is the bandwidth measured at the master process. [Reuss00]. Many scientific applications use this scheme.
One nice feature to the *SKaMPI*-package is the report generator. After all tests have finished one can use the generator to make it analyze the test results and output a postscript document. This contains among the graphical prepared results of the tests also useful comments.
The developers of *SKaMPI* aim to collect a lot of results of their benchmark run on many different machines and architectures to build up a database. These information make comparisons possible, which in turn can be useful when developing applications.

PMB The *PMB*-package (Pallas MPI Benchmark) has been developed by Pallas GmbH, Germany. The package is freely available including the source code. It is a concise set of basic MPI benchmark tests. The current version 2.2 consists of two major parts: PMB-MPI1 (discussed here) and PMB-MPI2 which needs MPI-2 extentions. The benchmark tests are divided into three classes: single transfer (for instance *ping-pong*), parallel transfer (for instance

ping-ping) and collective operations. Within the first class only two processes are active and communicating with each other. With no background jobs these are optimal conditions to achieve high bandwith results. For more practical, application oriented results all processes are participating in communication within the second class. Finally, the third class contains MPI collective opearations. The *PMB*-package takes quite the opposite way as *SKaMPI* does. Its results are stored as raw timings and no interpretation or graphical presentation is performed. Detailed information about the benchmarking tests performed by *PMB* can be found within the documentation part MPI-1 [Palla00].

4.2.2 Profiling Tools

Jumpshot There is a number of profiling tools for postmortem performance analysis and visualization in this series. The development started with tools called *Upshot* and *Nupshot* and later on several versions of *Jumpshot*. The development was based on two issues. The visualization quality of the first versions was quite poor and so was the speed of the tools. Second reason was the increasing size and complexity of parallel applications that users wanted to profile. This led to large log files which slowed down the available tools even more. Now there are three log-file formats available. The standard is still the Common Log format *CLOG*, but it might soon be replaced by the scalable log file format *SLOG*, which can only be read by the latest version of the Java2-based visualization tool *Jumpshot-3*. More information about the history of the tools and the log file formats can be found at [Chan02].
Jumpshot-3, as all tools of this series, comes with the *MPICH*-package and is located within the source tree in `mpe/viewers/Jumpshot-3`. The MPE library is used to log the timings of the MPI routines.

Here is an example in which way profiling with *Jumpshot-3* can be done. The application used is `mpptest`. The profiling shall take place using MPICH-G2 and Globus2. It is convenient to configure and build the *perftest*-package as usual and just relink a MPE-enabled version of the binary.

```
bash-2.05$ mpicc -mpilog -O3 -o mpptest-mpe mpptest.o gopf.o grph.o ovlap.o pair.o \
           pattern.o getopts.o rate.o mpe_seq.o copy.o halo.o -lm
```

The format in which the logging data shall be stored can be set by an environment variable, which can be provided to the processes for a Globus2 job within the RSL script. The script can easily be generated.

```
bash-2.05$ mpirun -np 2 -dumprsl mpptest-mpe -quick -fname mpe-test >mpe-test.rsl
```

Next step is to readjust the RSL script to be read as follows.

```
+
( &(resourceManagerContact="jack1.oscar")
   (count=  1)
   (label="subjob 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
        (MPE_LOG_FORMAT SLOG)
                (LD_LIBRARY_PATH /ra/projects/packages/globus-2.0/globus/lib/))
   (arguments= "-quick" "-fname" "mpe-test")
   (directory="/home/ra/regra/perftest-1.2-g2-mpe")
```

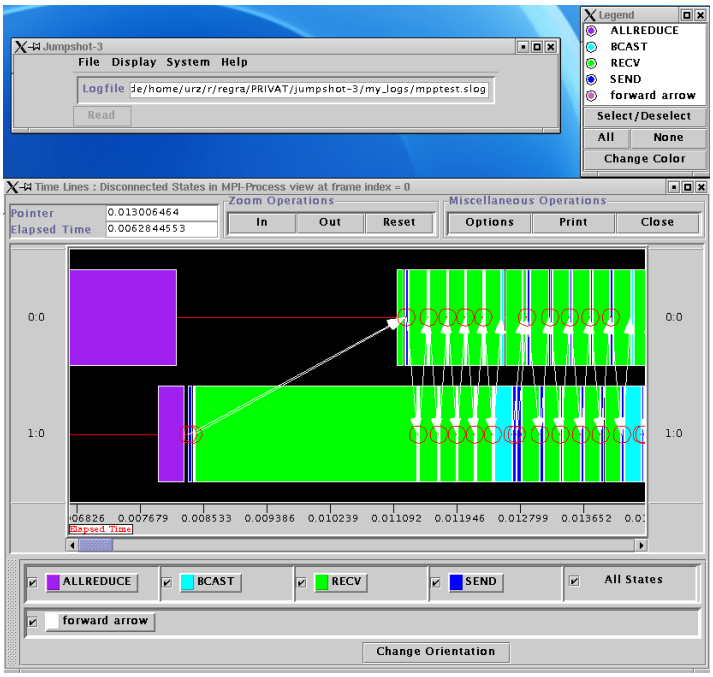


Figure 3: *Jumpshot-3* visualizing roundtrip pattern

```
   (executable="/home/ra/regra/perftest-1.2-g2-mpe/mpptest")
)
( &(resourceManagerContact="jack2.oscar")
   (count=  1)
   (label="subjob 1")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
        (MPE_LOG_FORMAT SLOG)
                (LD_LIBRARY_PATH /ra/projects/packages/globus-2.0/globus/lib/))
   (arguments= "-quick" "-fname" "mpe-test")
   (directory="/home/ra/regra/perftest-1.2-g2-mpe")
   (executable="/home/ra/regra/perftest-1.2-g2-mpe/mpptest")
)
```

Finally, the job can be submitted by supplying the name of the RSL script to `mpirun`. Automatically a file called `mpptest-mpe.slog` is generated. To examine the profiling data, *Jumpshot-3* needs to be configured and built by invoking, as usually, `configure` and `make`. That goes straightforward if there is an appropriate version of Java installed on the system. *Jumpshot-3* can now be used to open the log file and trace the communication between both of the processes of `mpptest`. This can be seen as one possible way to prove if the default communication pattern used by the tool is really the roundtrip pattern. As one can see in figure 3 this is true. After an phase of initialization both processes exchange messages in a ping-pong manner. An interesting fact is the time that had been spent in sending (blue) and receiving (green) MPI operations. In most cases the blue bar is too narrow and can hardly be perceived.

5 Performing Benchmark Tests

The previous section explained among others the benchmarking tools `mpptest` and `goptest` from the *perftest*-package. They are used here to perform a test suite, which is splitted into three test levels and described within the following episodes:

- Machine or SMP Level
- Cluster or LAN Level
- Cluster-of-Clusters or CoC Level

But, first of all, the suite of benchmarking tests needs to be explained. The denominations used are derived from the *perftest*-package's documentation and source code.

5.1 Benchmarking Suite

The benchmarking suite presented here will be used in all test levels with slight modifications due to various number of used processors. The test suite is based on the capabilities of `mpptest` and `goptest`, some comparisons and hints to other benchmark tools are given, too. Not all of the results of the tests that have been performed can be presented within this study on the following pages. A few representative ones had to be chosen.

5.1.1 Point-to-Point Test Cases

In the case of point-to-point communication the suite contains four test cases. Each of these in a blocking (or synchron) and a non-blocking (or asynchron) version. As 'blocking' refers to the state of the buffer, not whether the message has been delivered.

➟ *Round-Trip*
The round-trip pattern uses two processes for communication. The first process, also named the master process, sends one message of *size* bytes to the other process, also named slave process. Before the send function is called a timestamp is saved as t_0. After the slave has received the message it sends one message of same size back to the master. When having completely received the message a timestamp t_1 is saved. See figure 4. The elapsed time $\Delta t = t_1 - t_0$ is the round-trip time. Written to the output is the 'one way value', which is estimated by $\frac{\Delta t}{2}$. The bandwidth is accumulated for incoming and outgoing messages. This test is repeated several times with the same message size before the value of *size* is increased.
The Pallas MPI Benchmark names this test *ping-pong*, its timing result is also $\frac{\Delta t}{2}$. It is directly comparable to `mpptest`'s round-trip test in blocking mode.

➟ *Bisection*
This test performs a maximum of communication between all processes. During the biscetion test the number of processes is divided into two equally sized parts. Each process of the one half communicates with one process of the other half. The basis of all communication is the round-trip pattern. This test is interesting as it will help to

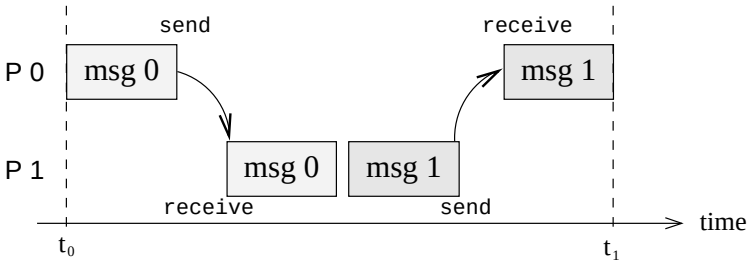


Figure 4: Round-Trip pattern

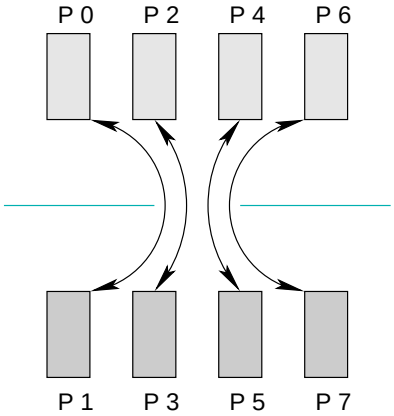


Figure 5: Bisection test case

find the bottleneck within the communication network - if one exists - as the number of processes or nodes participating is increased. The resulting bandwidth in the output graph is the measured aggregated bandwidth. If all nodes involved have the same interface bandwidth bw_{nodes} the optimum would be $\frac{NP*bw_{nodes}}{2}$. Obviously, the bisection test can only be run on even numbers of nodes.

➟ *Head-to-Head*
This pattern uses two processes for communication. After an initial synchronization the process having the master rank saves the start timestamp t_0. Each of both processes sends one message to the other one. After the receipt of the message the master saves the second timestamp t_1 and calculates the elapsed time $\Delta t = t_1 - t_0$. See figure 6. The Pallas MPI Benchmark names this test *ping-ping*, as no reply is sent to either of the messages. What has to be concerned when comparing the results of `mpptest`'s head-to-head in non-blocking mode to PMB's ping-ping (which also uses non-blocking MPI-calls) is following: The elapsed time written to output is $\frac{\Delta t}{2}$ in case of `mpptest` and Δt respectively.

➟ *Halo Exchange* In many applications data partitioning makes it necessary to introduce 'halo' or 'ghost' cells at the interface boundaries. These are used as a kind of cache and allow, for instance, derivatives and other operations to be performed on local vari-

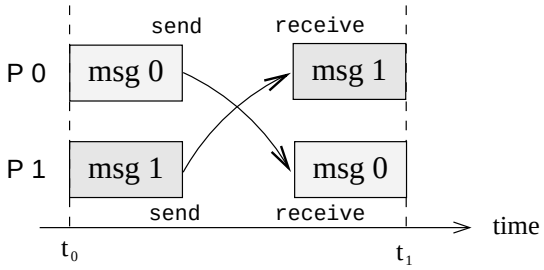


Figure 6: Head-to-head pattern

ables only. The number or thickness of halo cells can be from one to tens or hundreds depending on the operations and partitioning stencil. In the example in figure 7 two halo cells are used. Halo cells are updated as required, which is the basis of the halo exchange test.

In each of these test cases the results are achieved in a series of tests, measuring the available effective bandwidth and latency as a function of message size. Thus, various message sizes from 0 to 256 kbytes are used. Table 1 shows an overview.
The advantage of segmenting the test this way is obvious. Up to the lower maximum message size a small increment is used to obtain a fine grained result. For larger maximum message sizes a higher increment is sufficient. The tests are repeated four times to get a more precise result, as random influences on the network and on processor load, for instance by non-periodic time consuming interrupt handlers, can be filtered out. One can leave this task to the benchmarking tools by applying `-reps 4` as argument on the command line. The column headed 'auto dx' lists the values used for the parameter of the same name. In conjunction with the parameter `-auto` the tool `mpptest` tries to generate smooth graphs by calculating intermediate message sizes itself, which are at least that amount larger than the previous size that is given by `auto dx`.
To avoid influences of caches, especially CPU caches, the argument `-cachesize` is applied to `mpptest`'s command line. It makes the tool create a buffer of given size, in this case nearly 1 Mbyte, which is read and/or written sequentially in a ring buffer manner, independently of the current message size.

5.1.2 Collective Test Cases

The suite of collective operations makes use of three test cases, which are mostly used in many MPI applications. The tests (except the synchronization test) were run using various message sizes to get information about the scalability as a function of message size, and on different number of processors/nodes to achieve information about the scalability as a function of the number of participating processes. The communicator used in all MPI operations is `MPI_COMM_WORLD`. As in all point-to-point test cases the tests were also repeated four times.

- *Broadcast*

benchmark	max. message size	step width	auto dx
Latency	50	1	1
Bandwidth 2	2048 (2 kbyte)	16	8
Bandwidth 8	8192 (8 kbyte)	64	32
Bandwidth 16	16384 (16 kbyte)	128	64
Bandwidth 64	65536 (64 kbyte)	256	128
Bandwidth 256	262144 (256 kbyte)	4096	2048

Table 1: Used message sizes

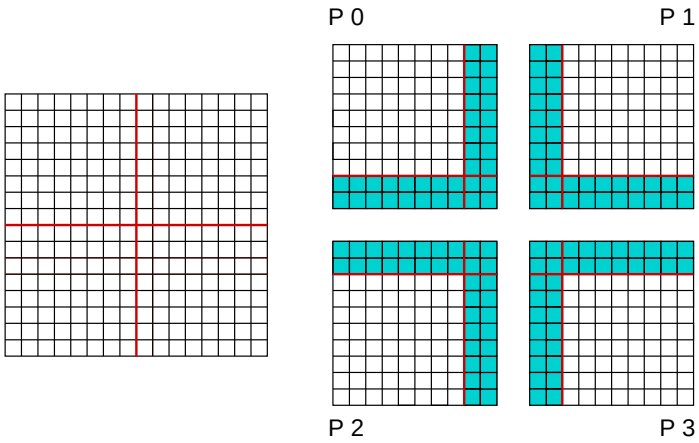


Figure 7: Data partitioning with halo cells

The broadcast test, also known as scatter test, performs broadcast operations of messages with given size on every process. The result is the average of each process' elapsed time.

- *Integer Reduction*
  The integer reduction test performs an `MPI_Allreduce` on each process on an integer buffer of a given message size.

- *Synchronization*
  The barrier or synchronization test makes a synchronization point (rendevous) between all involved processes. The test consists of a loop of `MPI_Barrier`-call.

An example of the script that has been used to run the benchmark suite can be found in the appendix A on page 41. In each test case presented above the performance of MPICH-G2 is compared to another MPICH-device. At the end of each of the following episodes the test environment is explained briefly and a summarizing table of all tests that had been performed is given. A short conclusion which evaluates the results closes each episode.

5.2 Tests on Machine Level

This benchmarking test will yield to the realization that MPICH-G2 does not support message passing using a fast shared memory communication method for SMP machines. This causes intra-machine communication between two or more processes to be done over TCP. The overhead of the protocol stack leads to poorer benchmark result in both cases: latencies and bandwidth. To illustrate the issue a comparison will be made. The MPICH version supporting shared memory is configured with `-with-device=ch_shmem -with-comm=shared`. This triggers to build the package with the device *ch_shmem*, which realizes message passing over System V Inter Process Communication (IPC). The opponent version is MPICH-G2, which is MPICH and the *globus2*-device. Its configuration and installation are explained in detail in section 3.2 on page 13 except one difference: `-with-comm=shared` is additionally

passed to `configure`. The MPI performance test tool `mpptest` is configured and compiled once using MPICH with device *ch_shmem* and once using MPICH-G2 compiler, linker and library.

This test is not intended to compare apples and oranges as described in section 4.1.3, but to prove the lack of shared memory support within the *globus2*-device and to illustrate the overhead caused by the TCP-protocol stack even if communication takes place within one machine. Furthermore the shared memory communication is still a message passing implementation and makes also use of synchronization as others do.

5.2.1 Results

The most important results of the latency and bandwidth measurements are shown in figures 8 to 11. All figures can be found in the appendix B.

To achieve a consistent and complete series of performance measurements the set of collective MPI operation tests has also been performed on system level using once the *ch_shmem*-device and once the *globus2*-device.
The synchronization test cannot be presented graphically as it is not a function of message size. The results are as follows:

globus2	82.016 µsec
ch_shmem	11.504 µsec

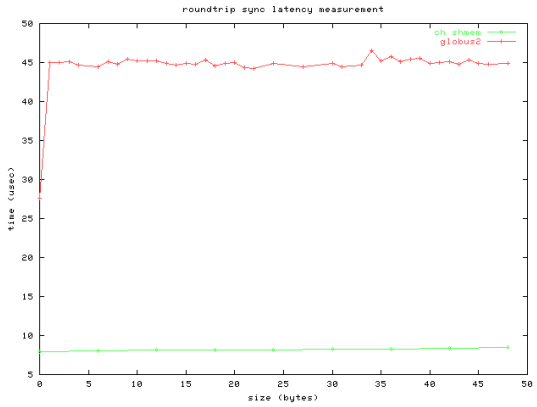
The difference is obvious. Again, due to the lack of shared memory support within the *globus2* device, the synchronization of both processes takes almost eight times longer.

5.2.2 Summary

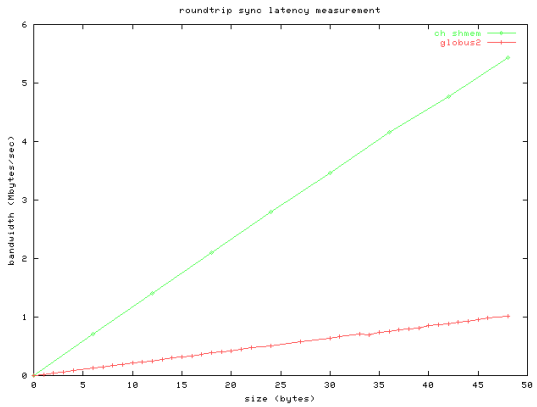
One can determine that the latency differences between the *ch_shmem*- and the *globus2*-device cannot be neglected. For small packet sizes the differences are almost constant (for instance see figure 8). At larger packet sizes the *ch_shmem*-device can even improve its behavior. Most of the overhead in MPICH-G2 is introduced by the TCP-protocol stack, which might also be the main reason for the increase of the timing differences due to higher amount of flow control and CPU utilization at larger message sizes. Within the set of collective tests, the synchronization test shows most obvious the timing differences of both implementations.

These tests were performed on one dual processor machine with two AMD Athlon MP 1600+ (1400 MHz real frequency) on a Tyan Tiger MPX S2466N-4M motherboard with AMD 760MPX chipset, 512 Mbytes DDR PC266 CL2 unbuffered memory.

Table 5.2.2 shows an overview of all tests that have been performed on system level.
Notice: During the head-to-head test `mpptest` did not terminate in time for message sizes larger than approximately 125kbytes.

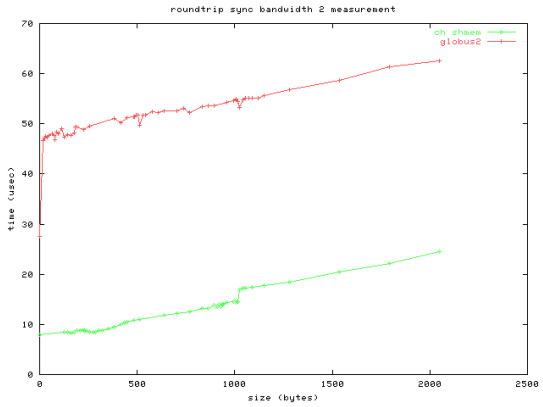


(a) Timings

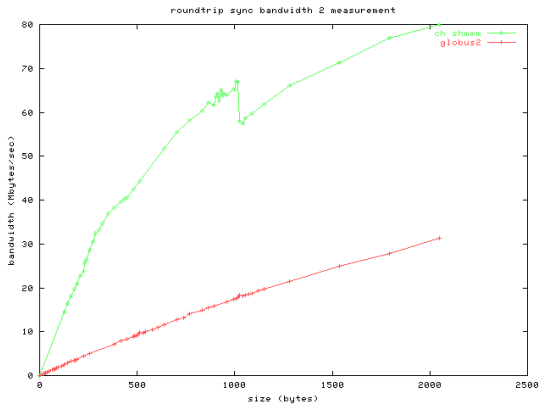


(b) Bandwidth

Figure 8: Point-to-point latency (roundtrip, blocking), maximum message size: 50 bytes

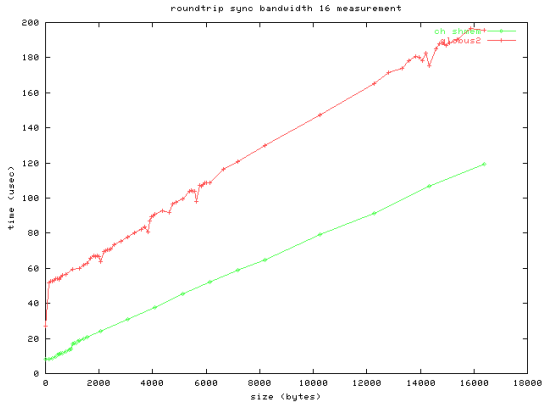


(a) Timings

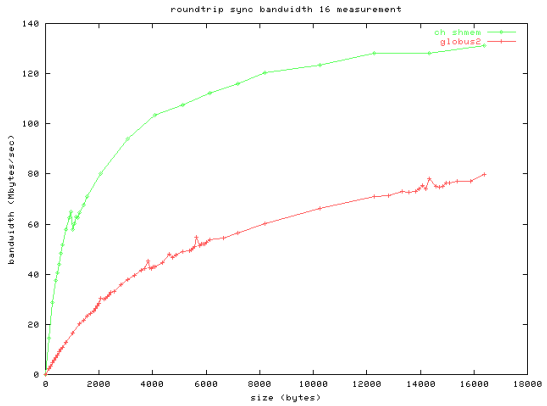


(b) Bandwidth

Figure 9: Point-to-point communication (roundtrip, blocking), maximum message size: 2 kbytes

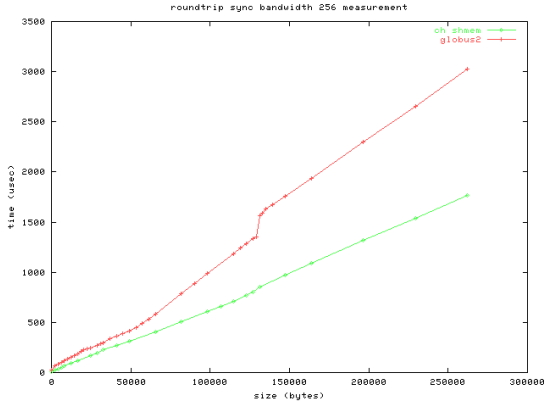


(a) Timings

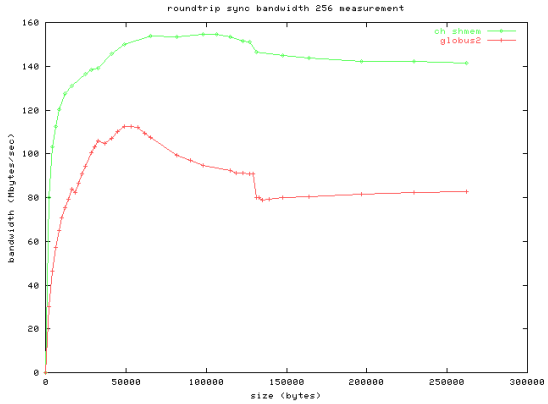


(b) Bandwidth

Figure 10: Point-to-point communication (roundtrip, blocking), maximum message size: 16 kbytes



(a) Timings



(b) Bandwidth

Figure 11: Point-to-point communication (roundtrip, blocking), maximum message size: 256 kbytes

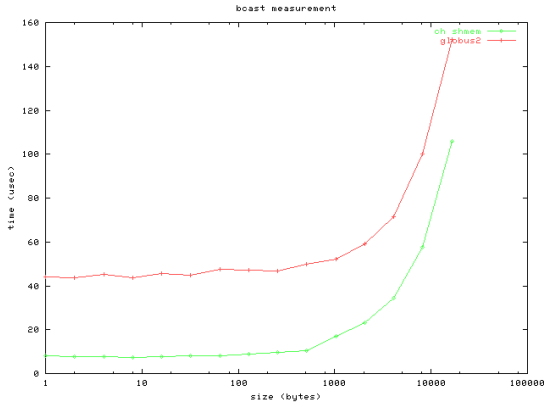


Figure 12: Collective operation: broadcast, maximum message size: 16 kbytes

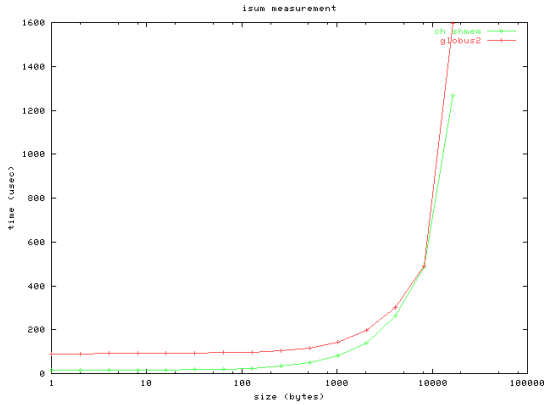


Figure 13: Collective operation: integer reduction, maximum message size: 16 kbytes

	mode	np	roundtrip	bisection	halo exchange	head-to-head	collective
globus2	sync	2	✓	-	-	✓	✓
	async	2	✓	-	✓	✓	
ch_shmem	sync	2	✓	-	-	✓	✓
	async	2	✓	-	✓	✓	

Table 2: Summary of tests performed on system level

5.2.3 Conclusion

The tests and the results presented here validate that shared memory support in machines with symmetric multi processing is not available in the current release of MPICH-G2, although the appropriate argument is known to the `configure` tool. This might just be an inheritance of the general version of MPICH from which it is derived. The MPICH-G2 homepage [MG2Te02] says in the section of planned features for future releases that there will be shared memory support some time. Especially when using MPICH-G2 in institutional or local cluster-of-clusters systems shared memory support is a very important feature as multiprocessor systems are very common in clusters nowadays.

5.3 Tests on Cluster Level

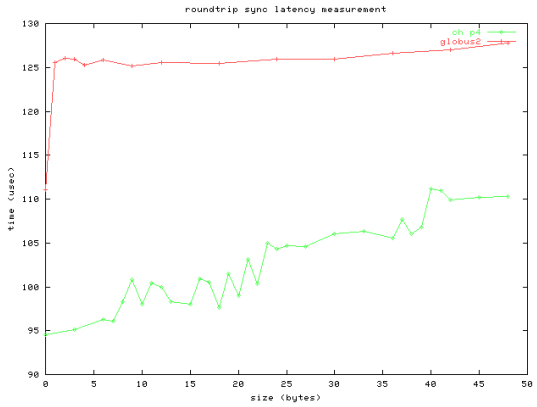
This section presents the results of the benchmark suite presented in section 5.1 on page 21 run on LAN or cluster level. All nodes that participate in the test are connected over Fast Ethernet and plugged into one switch. This is the typical environment in which MPICH with the *ch_p4*-device is used. That is why the performance measurements presented here compare it to the *globus2*-device. Similar to the test on SMP level, `mpptest` is configured and compiled once using MPICH with device *ch_p4* and once with *globus2* and linked with the appropriate linker and library.

The aim is to find out in which way the enhancements within the *globus2*-device, which enable it for grid computing, influence its the performance in a non grid environment. From the results of the tests above one presumption could be suggested: the grid-enabled device might again have higher latencies than its counterpart.

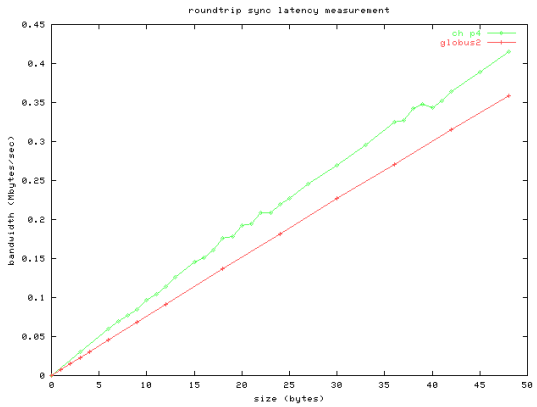
As mentioned earlier, only a few representative graphs of results procured can be presented here. Table 3 on page 34 gives an overview of all tests perfomed, and appendix B.1.

5.3.1 Point-to-Point Results of two Processes

The point-to-point part of the test suite had been repeated on two, four, eight and 16 nodes (each with the same number of participating processes).
The tests using two processes running on two different nodes illustrate the latency issues very well. At smaller message sizes the *globus2*-device has a higher latency of approximately 20 to 30 µsec (see figure 14 on the next page). With an increasing message size, the latency difference decreases rapidly and can be seen as void from 300 bytes upwards. The slight perturbation near 1500 bytes is likely due to a protocol change within the *MPICH* implementation. In figure 16 on the following page one can see both graphs printed above each other. So for larger message sizes there is no disadvantage in communication performance for the *globus2*-device.

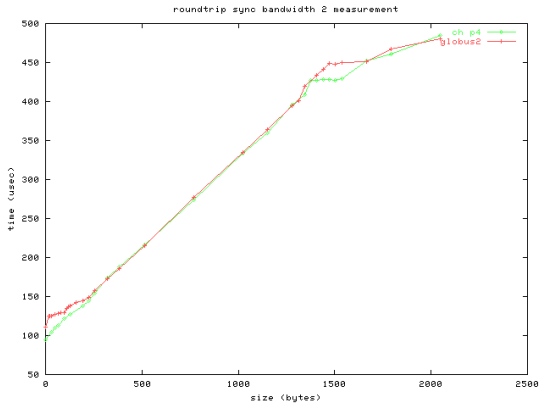


(a) Timings

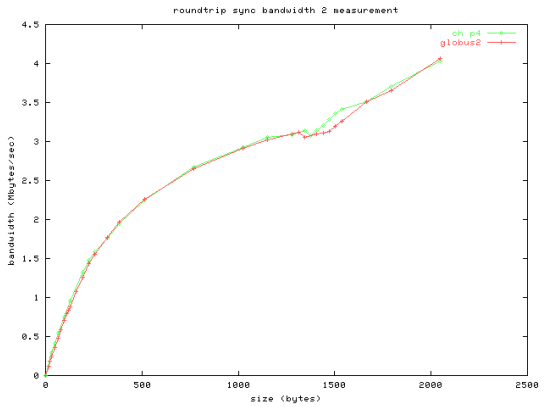


(b) Bandwidth

Figure 14: Point-to-point latency (roundtrip, blocking), maximum message size: 50 bytes

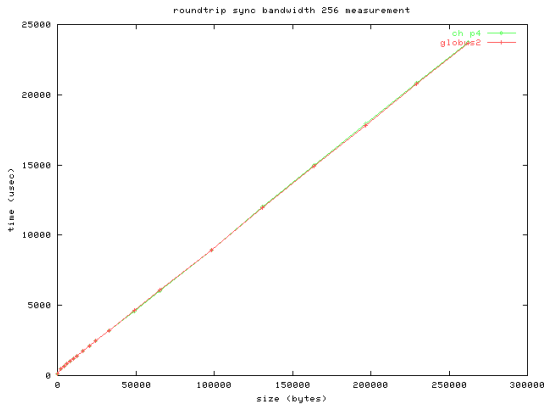


(a) Timings

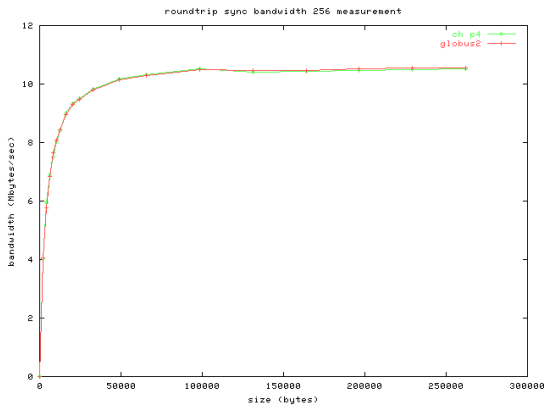


(b) Bandwidth

Figure 15: Point-to-point communication (roundtrip, blocking), maximum message size: 2 kbytes



(a) Timings



(b) Bandwidth

Figure 16: Point-to-point communication (roundtrip, blocking), maximum message size: 256 kbytes

5.3.2 Point-to-Point Results of four Processes

Here it is the first time that more than two processes take part in one test. So some sinlge tests have to be changed. For instance, the simple roundtrip test cannot make use of more than two partners. Instead the bisection test is used. Further on it is similar to the head-to-head test, which is to be skipped onwards. But in spite of that the halo-exchange test can, of course, make use of more than two processes communicating. It comes more and more into the focus on the following sections.

The bisection test shows the same behavior when message sizes increase as the roundtrip test with two processes. The major difference is the maximum gained bisection bandwidth of more than 22 Mbyte per second.
The result of the halo-exchange test is quite different and the reason for that is not comprehensible for now. Both implementations start at smaller message sizes quite normal, but at message sizes larger than approximately 10 kbytes the *ch_p4*-device shows a strange behavior. Elapsed time grows much faster than the time of its counterpart. Figure 17 illustrates the result.

5.3.3 Point-to-Point Results of eight Processes

In this part of the test there is no special occurance. The bisection bandwidth scaled well with the number of participating processes, it reached more than 44 Mbytes per second. The halo exchange pattern led to a quite similar graph as in the previous section.

5.3.4 Point-to-Point Results of 16 Processes

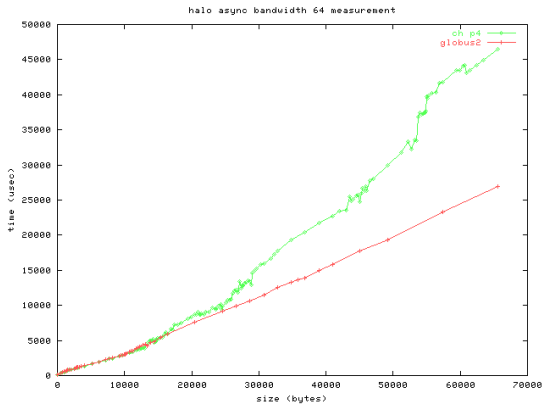
These are the results achieved using all available 16 nodes of the cluster in the tests of point-to-point communication. Again, as one could presume, bisection bandwidth scaled perfectly with the increased number of processes. The maximum bandwidth reached more than 88 Mbytes per second as shown in figure 5.3.4 on the next page.

5.3.5 Collective Results

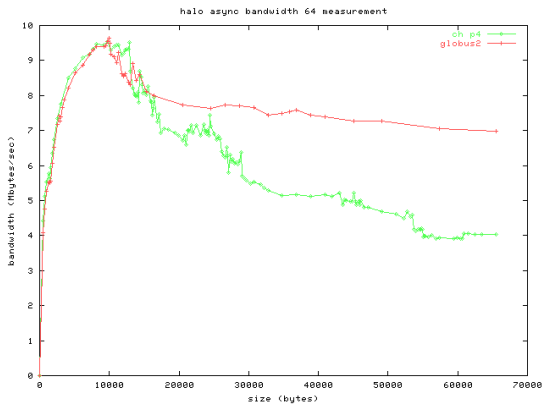
The collective operation tests were run from two up to 16 processes on the homogeneous cluster. Comparing both devices with each other and also to the point-to-point test results one can say both show a typical behavior here. At small message sizes the *globus2*-device is often slightly slower than the *ch_p4* one. With increased message sizes both implementation perform nearly equally during the broadcast test, in spite of relatively high variations. The integer reduction shows a different view. At smaller message sizes both are about the same and at increased messages *globus2*-device performs poorer than its counterpart, see figures 21 and 22.

5.3.6 Summary

All of the tests in this section of a pure, homogeneous intra-connected cluster system, lead to a similar results: *ch_p4* is much faster for small messages, at larger message sizes both rivals

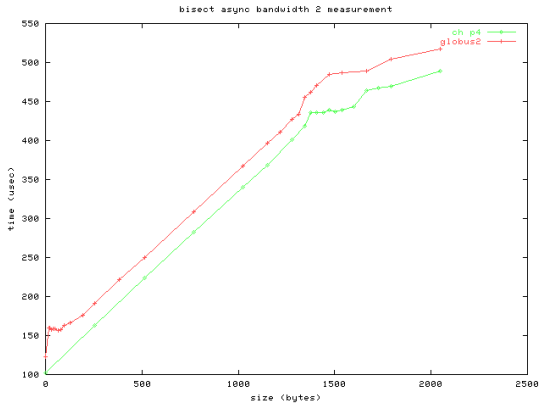


(a) Timings

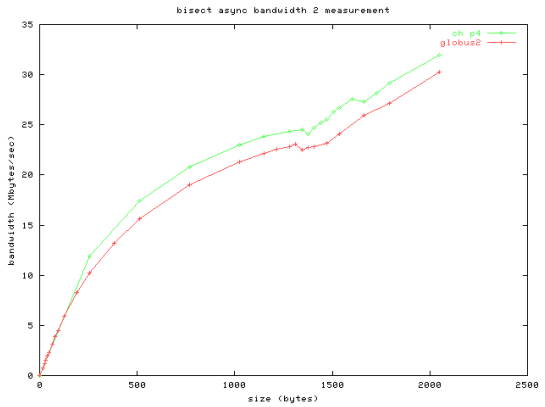


(b) Bandwidth

Figure 17: Point-to-point communication (halo, non-blocking), maximum message size: 256 kbytes

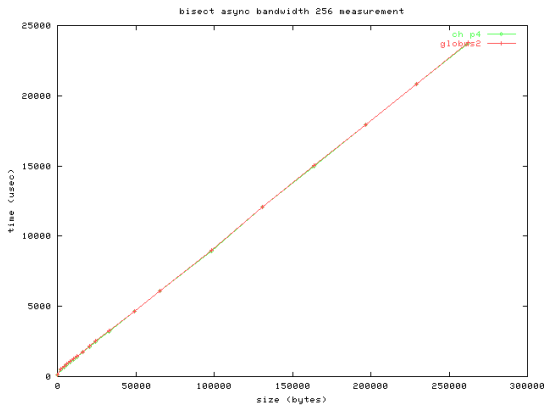


(a) Timings

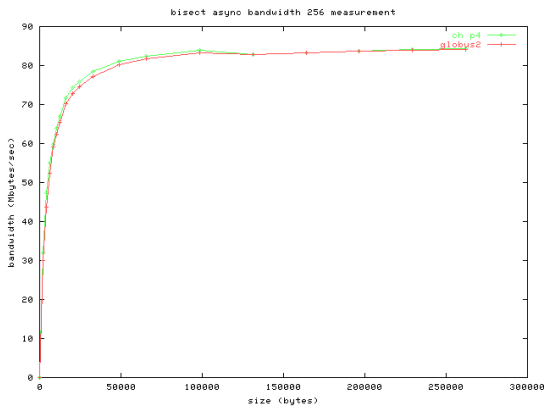


(b) Bandwidth

Figure 18: Point-to-point communication (bisection, non-blocking), maximum message size: 2 kbytes

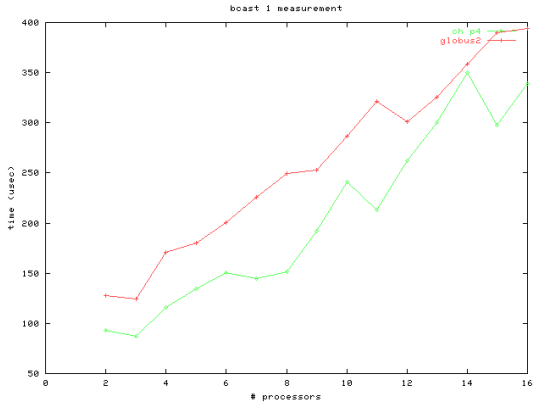


(a) Timings

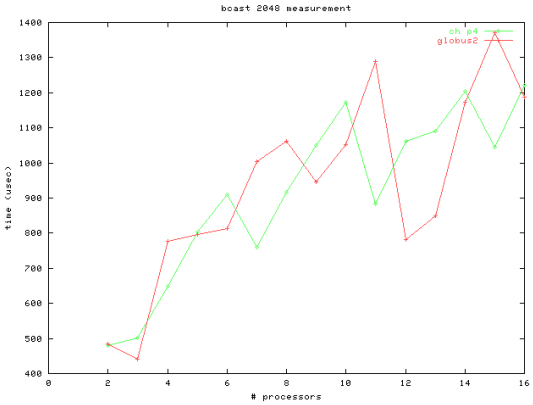


(b) Bandwidth

Figure 19: Point-to-point communication (bisection, non-blocking), maximum message size: 256 kbytes

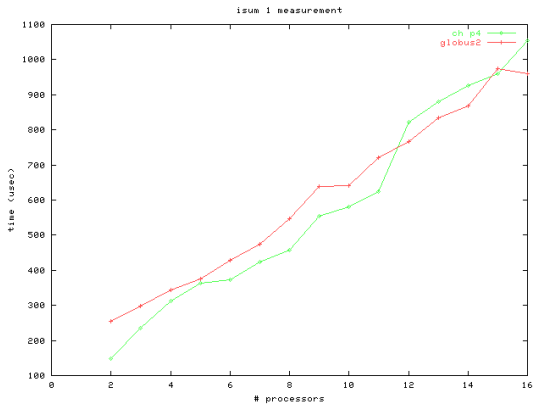


(a) message size 1 byte

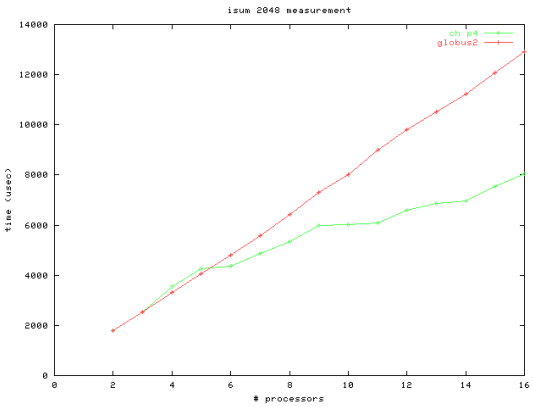


(b) message size 2 kbytes

Figure 20: Collective opearation: broadcast



(a) message size 1 byte



(b) message size 2 kbytes

Figure 21: Collective operation: integer reduction

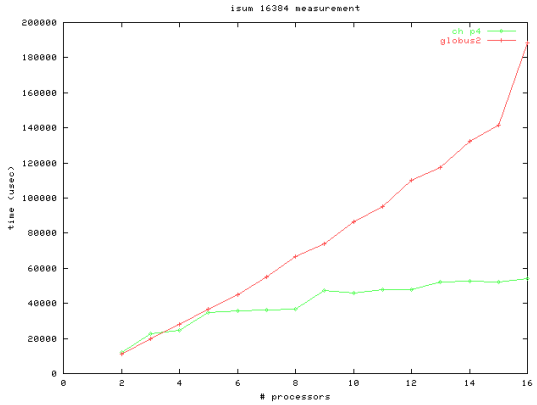


Figure 22: Collective operation: integer reduction, message size 16 kbytes

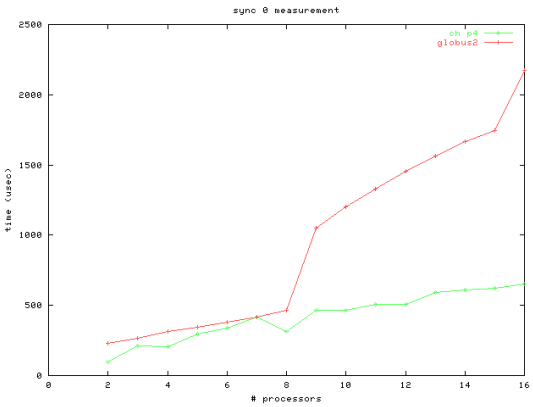


Figure 23: Collective operation: synchronization

are performing nearly equally.

All tests using one to eight processes were performed on dual processor machines with only one job on each node. The machines are equipped with AMD Athlon MP 1600+ (1400 MHz real frequency), on a Tyan Tiger MPX S2466N-4M motherboard with AMD 760MPX chipset, 512 Mbytes DDR PC266 CL2 unbuffered memory and onboard 3Com 3c905C 100 MBit network interface. Tests with more than eight processes additionally used Intel Pentium II (0.25µm Deschutes) 333 MHz on an Asus P2L97-DS motherboard with Intel 440LX chipset, 256 Mbytes SDRAM PC-66 and Intel PRO/100+ (Intel 82557-based 10/100 Ethernet PCI Adapter) 100 MBit network interface. All machines were connected to the same 3Com SuperStack II 3900 switch.

It is to mention that all machines have a diskless setup and the administration and the communication network is shared. Without further research it can hardly be described how this had influenced the tests.

Table 3 shows an overview of all tests.

Notice: During the head-to-head test `mpptest` did not terminate in time for message sizes larger than approximately 125kbytes.

5.3.7 Conclusion

All tests presented in this part of the LAN or cluster level give a good view on the performance of the *globus2*-device. The additional functionality to grid-enable the MPICH device, which is not needed or used here, has influences of the performance behavior in comparison to the *ch_p4*-device. But these influences are very few as they only ocure when small messages are sent. It is almost the same in both point-to-point and collective operations. So, on the one hand, you can state that for (local) homogeneous clusters it is not an advantage to use MPICH-G2. But on the other hand, there is no huge performance loss, if the application not only sends lots of small packets, when the *globus2*-device is use. Just the start-up time of the jobs is higher compared to MPICH and the *ch_p4*-device and can take up to several seconds.

5.4 Tests on Cluster-of-Clusters Level

The last section tries to find out if MPICH-G2 can play off its strength in a system which is combined of fast intra-connected subsystems, but with a relatively slow inter-connection link. Figure 24 on the following page illustrates the structure of this cluster-of-clusters system. The basis for the test is, again, TCP/IP over ethernet, since TCP/IP is the basis for all communication that is initiated by Globus components. There is no native support for other, high performance networks such as SCI or Myrinet. The possibility to use these System Area Networks (SANs) directly and efficient as part of the Globus resources would be a great enhancement in the development of cluster-of-clusters systems. There are two obvious ways. First is to make fast SANs useable over TCP/IP. This would be the poorer choice as most advantages, for instance such as very low latencies of these networks, were destroyed by a conventional TCP/IP software stack. The second way could be the usage of Globus' native or vendor supplied MPI interface. The problem in this case is following. The *globus2*-device is based on MPICH as most of the MPI-implementations for SANs are based on MPICH, too. This leads to a for now inevitable symbol clash during the link process. Although when

	mode	np	roundtrip	bisection	halo exchange	head-to-head	collective
globus2	sync	2	✓	-	-	✓	✓
	async	2	✓	-	✓	✓	
	sync	4	-	✓	-	-	✓
	async	4	-	✓	✓	-	
	sync	8	-	✓	-	-	✓
	async	8	-	✓	✓	-	
	sync	16	-	✓	-	-	✓
	async	16	-	✓	✓	-	
ch_p4	sync	2	✓	-	-	✓	✓
	async	2	✓	-	✓	✓	
	sync	4	-	✓	-	-	✓
	async	4	-	✓	✓	-	
	sync	8	-	✓	-	-	✓
	async	8	-	✓	✓	-	
	sync	16	-	✓	-	-	✓
	async	16	-	✓	-	-	

Table 3: Summary of tests performed on cluster level

configuring for the usage with a vendor supplied MPI library the symbols of the MPI standard get renamed, there are several non-standard symbols changing very frequently. This issue is explained more in detail in section *How does MPICH-G2 work?* in [MG2Te02]. The developers of MPICH-G2 hope to find a better solution to rename non-MPI MPICH symbols and to open the door for 'vendor supplied' Linux cluster support.

The tests presented in this section are based on the collective part of the test suite as only collective MPI operations within MPICH-G2 are topology-aware. And, further on, measuring bisection bandwidth would lead to the obvious result that the slower inter-connection link is the whole system's bottleneck. Two types of tests had been performed, first using 16 single processor nodes and second using 16 dual processor nodes. The results are given in the following sections.

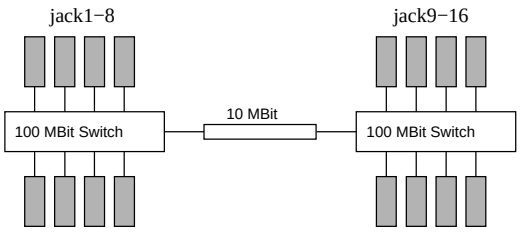


Figure 24: Structure of the CoC System

5.4.1 Collective Results of 16 Processes

This test makes use of only one processor per node, as all other tests before did. Three tests are compared with each other: *ch_p4*-device and *globus2*-device as ever, and additionally the *globus2*-device which gets information about the structure of the system. This is done by providing the environment variable `GLOBUS_LAN_ID` for each job and set it to unique values for all jobs that belong to one LAN or sub cluster. MPICH-G2 now has information that there are two groups of nodes, which can communicate over the protocol level of *LAN-TCP* and just both groups of them use the protocol level of *WAN-TCP*. The variable `GLOBUS_LAN_ID` can easily be set within the RSL script as illustrated by the following example.

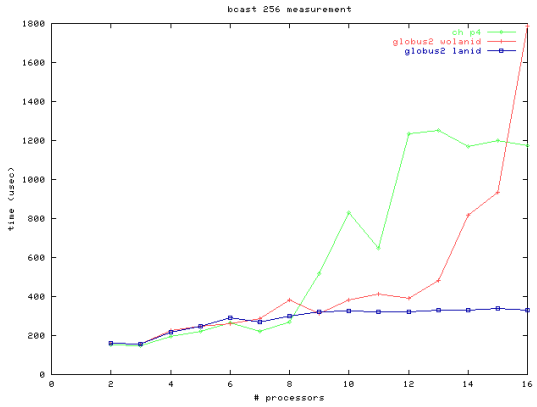
```
...
( &(resourceManagerContact="jack8.oscar")
   (count=1)
   (label="subjob 7")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 7)
                (GLOBUS_LAN_ID jacks1to8)
                (LD_LIBRARY_PATH /ra/projects/packages/globus-2.0/globus/lib/))
   (arguments= "-gnuplot" "-reps" "4" "-auto" "-fname" "bcast.mpl" "-bcast")
   (directory="/home/ra/regra/perftest-1.2-g2")
   (executable="/home/ra/regra/perftest-1.2-g2/goptest")
)
( &(resourceManagerContact="jack9.oscar")
   (count=1)
   (label="subjob 8")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 8)
                (GLOBUS_LAN_ID jacks9to16)
                (LD_LIBRARY_PATH /ra/projects/packages/globus-2.0/globus/lib/))
   (arguments= "-gnuplot" "-reps" "4" "-auto" "-fname" "bcast.mpl" "-bcast")
   (directory="/home/ra/regra/perftest-1.2-g2")
   (executable="/home/ra/regra/perftest-1.2-g2/goptest")
)
...
```

And indeed does the knowledge of the topology change the behavior of MPICH-G2. The broadcast operation takes much fewer time than without that information. This is realized as the implementation tries to use slow network links as early as possible. While in progress of the broadcast operation, the tree first spans to both sub-clusters and afterwards within each sub-cluster. The graphical results are shown in figure 25 on the next page.

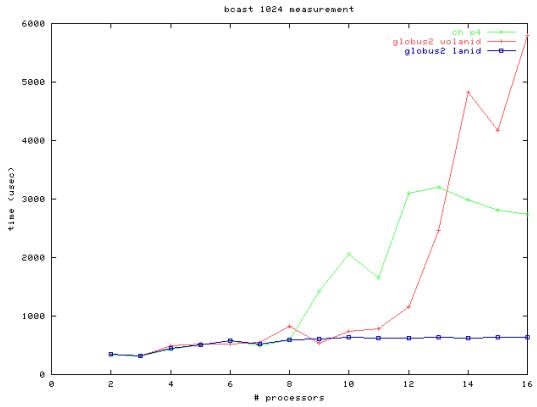
The results of the integer reduction test show a two edged picture in figure 26 on the following page. On the one side the *globus2*-device without having supplied topology information scales much poorer then its counterpart with that information. On the other side *ch_p4* is nearly as fast as *globus2* with that information of the topology. The answer is quite simple. According to [MG2Te02] MPICH-G2 does not yet provide a topology-aware implementation of `MPI_Allreduce`, but this function is used in *goptest*'s reduction benchmarks. MPICH-G2 uses in this case the default binomial tree algorithm of MPICH.

5.4.2 Collective Results of 32 Processes

The very last of all test now uses both processors on each node. Although the structure of the network remains the same as in the previous case, there is a change to the topology. Both processes on a node belong to one Globus subjob (triggered by `count=2` within the RSL

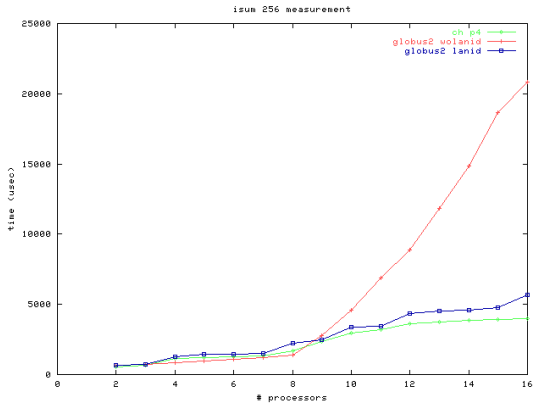


(a) message size 256 bytes

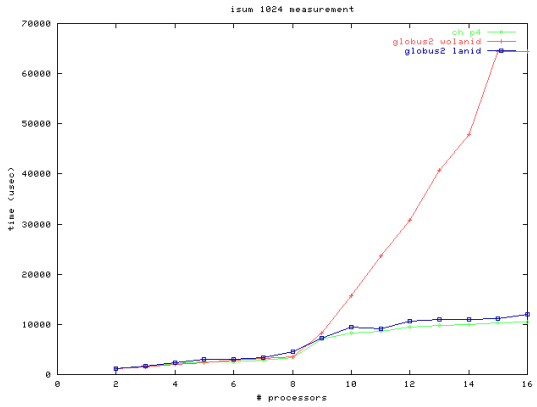


(b) message size 1 kbyte

Figure 25: Collective operation: broadcast

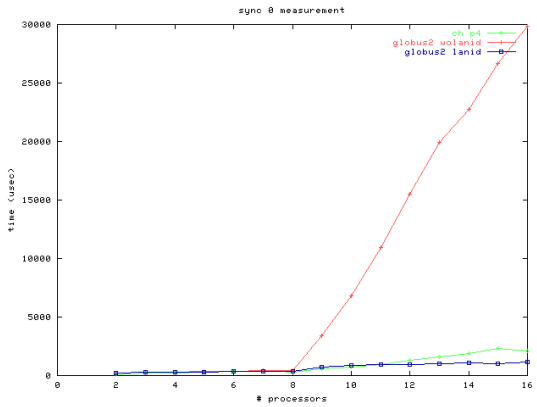


(a) message size 256 bytes

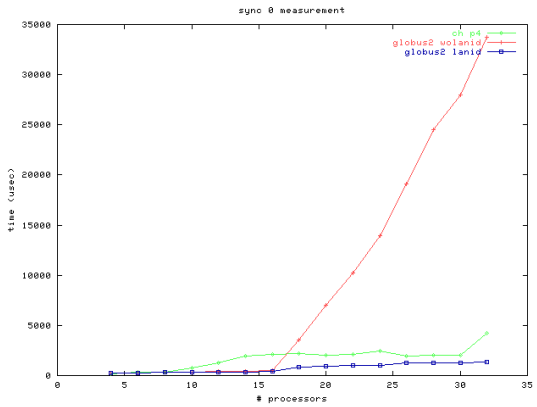


(b) message size 1 kbyte

Figure 26: Collective operation: integer reduction



(a) 16 processes



(b) 32 processes

Figure 27: Collective operation: synchronization

script). That enables a third protocol level of MPICH-G2 that is *intra-machine TCP*. The test were run on a per node basis, that is the overall number of participating processes was increased by two for each step so that at each time none or both processors were used.

The results shown in figures 28 and 29 are somehow compareable to the previous test. MPICH-G2 with supplied LAN-ID performs outstandingly well for the broadcast operation test.

5.4.3 Summary

The *globus2*-device can increase its communication performance in both cases if it has information about the topology. It performs even better when the level of intra-machine TCP is additionally used. MPICH-G2 can distinguish between processes on one machine and processes on different machines, MPICH with *ch_p4* cannot do.

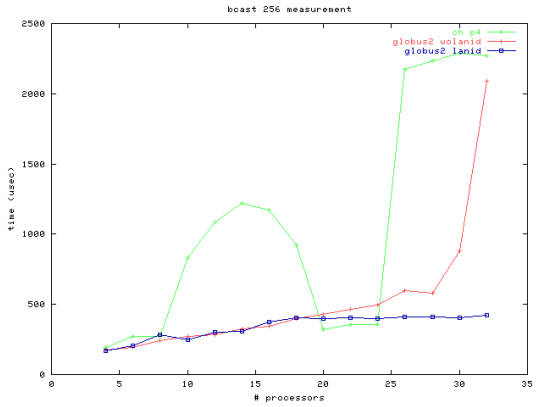
All tests we run on the same machines as in the cluster level tests before, see page 33. Only the network connections were changed to set up a system in a cluster-of-clusters manner. Two 3Com Superstack II switches, each connecting eight nodes, were itself connected by a 10 MBit repeater. It is to mention, again, that all machines have a diskless setup and the administration and the communication network is shared.
Notice: Collective operation tests used a shorter list of packet sizes: 0, 256, 512, 768, 1024 bytes.

5.4.4 Conclusion

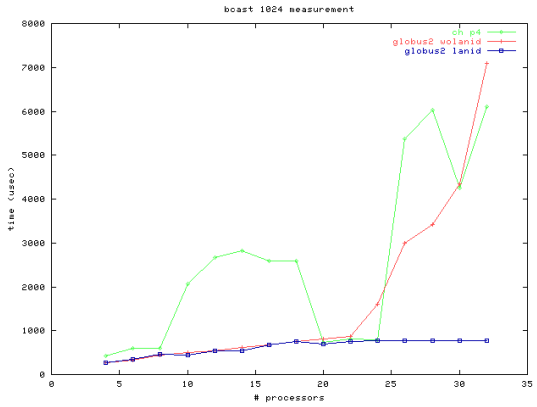
The Globus Toolkit has been developed for building large computational grids spreading all over the world. The last tests prove that there are also advantages when using grid technologies on less huge but also heterogeneous environments such as institutional and local cluster-of-clusters systems. In MPICH-G2 there are several collective MPI-operations, which are topology-aware and so have from slight up to an immense performance increase, for instance synchronization and broadcast. But there are also diadvantages, especially in point-to-point communication with small packet sizes. MPICH-G2 has often much higher latencies than other MPI libraries. Beyond this, the Globus Toolkit is large packet of software that needs to be installed and maintained and certificate management additionally causes effort. For small jobs, as most of the collective test done here, the start-up time of a Globus job is a

	mode	np	collective
globus2	with LANID	16	✓
	without LANID	16	✓
	with LANID	32	✓
	without LANID	32	✓
ch_p4	-	16	✓
	-	32	✓

Table 4: Summary of tests performed on cluster-of-clusters level

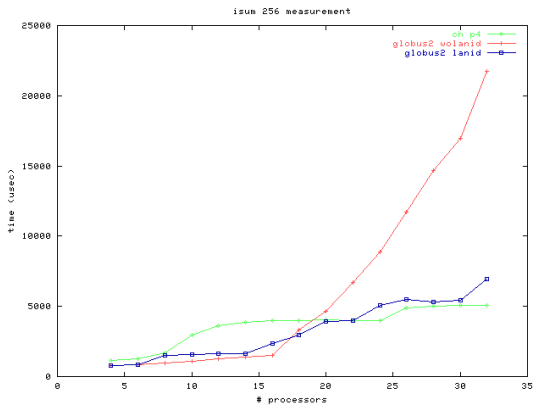


(a) message size 256 bytes

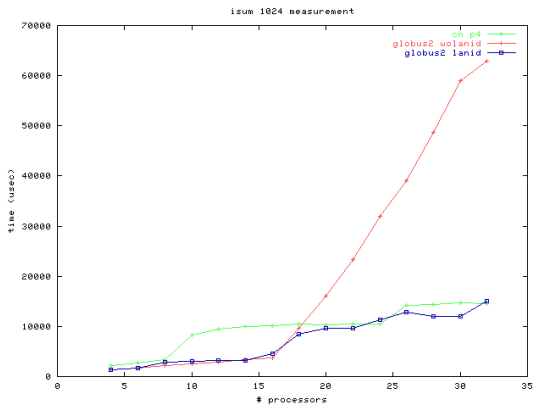


(b) message size 1 kbyte

Figure 28: Collective opearation: broadcast



(a) message size 256 bytes



(b) message size 1 kbyte

Figure 29: Collective opearation: integer reduction

disadvantage, too. But, in practice jobs normally run much longer so that the start-up makes neglectable delay.

One more important fact, which is also negative, is the lack of a MPI library, that can handle a cluster or pool of, for instance Linux PCs with various intra-connects, as one Globus resource that, in Globus' point of view, provides a vendor supplied implementation of MPI.

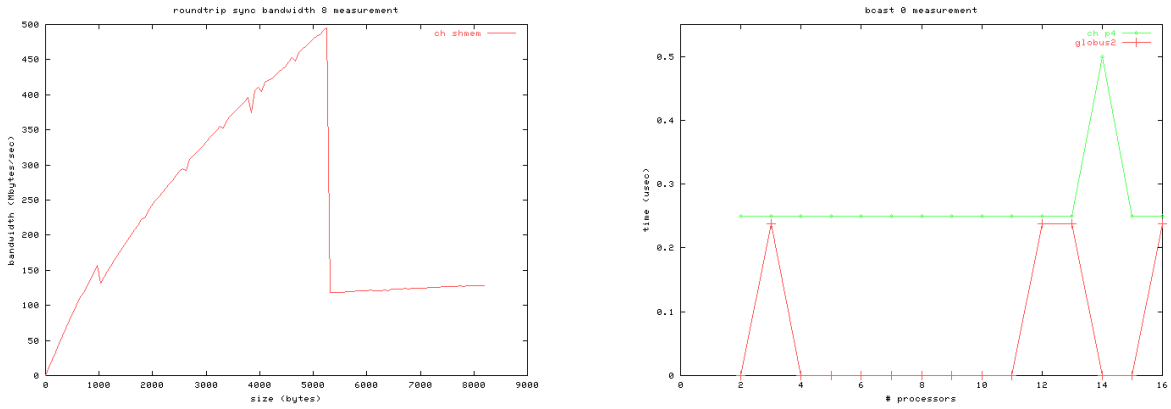
For SMP machines it would be appropriate to have shared memory support, either by a new MPICH-G2 communication level between vendor-supplied MPI and intra-machine TCP or by the possibility to use such a library as a 'vendor-supplied' one.
From my point of view, it makes sense to prefer the Globus Toolkit combined with MPICH-G2 for cluster-of-clusters systems, that have high differences between intra-connection link speed and inter-connection link speed. The more subclusters take part and the more heterogeneity grows the more is the need to go away from software providing cluster solutions to ones that provide grid solutions. The best would be a solution in between, which combines the advantages of both.

5.5 Experiences and Issues

This section describes in short some experiences made during the phase of gathering the test results presented above. The pitfalls and problems discussed in section 4.1 are not of theoretical nature. During the tests several of these issues really occured. Many of them can be avoided by the tools of the *perftest*-package by supplying appropritate arguments. For instance, influences of caches need to be regarded and get more important if the speed of the communication links increases. This had been oservated in the tests of machine level. Without explicit instruction to use a large buffer the bandwidth in the range of small messages overexceeded the physical bandwidth of the main memory. From larger messages sizes, that did not completly fit into the cache, the bandwidth broke down to a realistic level near the main memory bandwidth (see figure 30(a) on the next page). Cache effects can be avoided by applying the argument `-cachesize` to the command line of `mpptest`.

The issue of clock resolution could be found when performing a broadcast test with a null message. The MPI libraries seem to realize this function with a kind of no-operation. This takes very few time and so the results show the minimum resolution of the clock that is the basis for `MPI_Wtime()` in these MPI libraries on that particular architecture. Figure 30(b) illustrates this issue. And, there is something more to learn from this. Results are given in µsec by `mpptest`. The clock's resolution seems to be 0.25µsec or $2.5 \cdot 10^{-7}$seconds. This needs to be concerned when looking at the plain timing results given up to a precision of 10^{-12}.

Variations in the measuring results is an issue that is tried to be solved by the algorithms used in the tools of the *perftest*-package. Tests are repeated several times and the results are averaged. It is possible to instruct the tools to output the range of the timings measured to obtain information about the variations. That was not used in this study as the aim is to provide an overview of the achievable performance. It would be an easy task to modify and repeat a



(a) Cache effects

(b) Null broadcast

Figure 30: Special issues experienced during measurement

particular test to get fine grained results including variations for further studies.

The last problem that occurred cannot be solved by the benchmarking tools. It is the issue of influences caused by unrelated applications or jobs. In a cluster system that is, of course, used by several persons it is sometimes difficult to find appropriate time slots in which no other user jobs are running a the nodes that shall be used for the particular benchmark test. Disturbances can lead to the need to repeat the whole test suite for a particular test case, which is very time consuming, for instance it took quite more than three hours each time to finish point-to-point tests for one number of participating processes.

A Example of a Wrapper for `mpptest`

```
#!/bin/bash

# 8 single machines, mpich-g2

#set to zero to skip particular test
RUN_P2P=1; RUN_COL=0

PROJECT=jack1-8_globus2
NP=8
MPIBIN=/ra/projects/packages/globus-2.0/mpich-g2/bin
PERFTEST=/home/ra/regra/perftest-1.2-g2
OUTDIR=/home/ra/regra/perftest-1.2-g2/run/8nodes

#point-to-point pattern
outerpattern=( roundtrip bisect halo head )
innerpattern=( async sync )
#point-to-point suite
suite[0]="latency";       param[0]="-size 0 50 1 -autodx 1"
suite[1]="bandwidth_2"; param[1]="-size 0 2048 16 -autodx 8"
suite[2]="bandwidth_8"; param[2]="-size 0 8192 64 -autodx 32"
suite[3]="bandwidth_16"; param[3]="-size 0 16384 128 -autodx 64"
suite[4]="bandwidth_64"; param[4]="-size 0 65536 256 -autodx 128"
suite[5]="bandwidth_256";param[5]="-size 0 262144 4096 -autodx 2048"

pt2pt_allparam="-gnuplot -reps 4 -max_run_time 10800 -cachesize 1000000 -auto \
               -tgoal 0.1 -sample_reps 20 -n_stable 8 -npartner $NP"

#collective suite
csuite[0]="bcast";
cparam[0]="-bcast -sizelist 0,1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384"
csuite[1]="isum";
cparam[1]="-isum -sizelist 0,1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384"
csuite[2]="sync";
cparam[2]="-sync"
coll_allparam="-gnuplot -reps 4 -max_run_time 10800 -cachesize 1000000 -auto"

#FUNCTION
#run point-to-point tests
run_p2p () {

result="okay"; retval=0
for ((opat=0; opat < ${#outerpattern[
 for ((ipat=0; ipat < ${#innerpattern[
    #skip bisect for NP<=2, skip halo if sync, skip others if NP>2
    skip=0
    case "${outerpattern[opat]}" in
        bisect)
          [ $NP -le 2 ] && skip=1
          ;;
        halo)
          [ ${innerpattern[ipat]} == sync ] && skip=1 || skip=0
          ;;
        *)
          [ $NP -gt 2 ] && skip=1
```

```
    esac
    if [ $skip -eq 0 ]; then
      for ((index=0; index < ${#suite[
       echo -n "Pattern: ${outerpattern[opat]}, ${innerpattern[ipat]} starting test \
                $index: ${suite[index]}... "
       $MPIBIN/mpirun -np $NP $PERFTEST/mpptest $pt2pt_allparam \                    60
                -${outerpattern[opat]} -${innerpattern[ipat]} \
                -fname $OUTDIR/$PROJECT-${outerpattern[opat]}-${innerpattern[ipat]}-${suite[index]}.mpl \
                ${param[index]} || result="failed"; retval=1
       echo "...finished ${suite[index]}, test $result"
      done
    else echo "skipped ${outerpattern[opat]} ${innerpattern[ipat]} at NP==$NP"
    fi
 done
done
return $retval                                                                     70
}

#FUNCTION
#run collective tests
run_collective () {

result="okay"; retval=0
for ((proc=2; proc <= $NP; proc++)); do
 for ((index=0; index < ${#csuite[
  echo -n "starting test $index with $proc proc.: ${csuite[index]}..."            80
    $MPIBIN/mpirun -np $proc $PERFTEST/goptest $coll_allparam \
             -fname $OUTDIR/$PROJECT-np$proc-${csuite[index]}.mpl ${cparam[index]} \
             || result="failed"; retval=1
  echo "...finished ${csuite[index]}, test $result"
 done
done
return $retval
}

#MAIN                                                                               90
if [ $RUN_P2P -gt 0 ]
 then
 run_p2p
 retp2p=$?
 else echo "skipped point-to-point tests"
fi
if [ $RUN_COL -gt 0 ]
 then
 run_collective
 retcol=$?                                                                         100
 else echo "skipped collective tests"
fi
```

B All Test Results

B.1 Tests on System Level

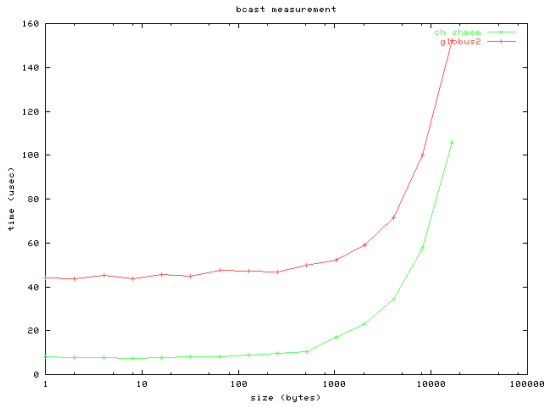


Figure 31: SMP level: Coll. oparation: broadcast

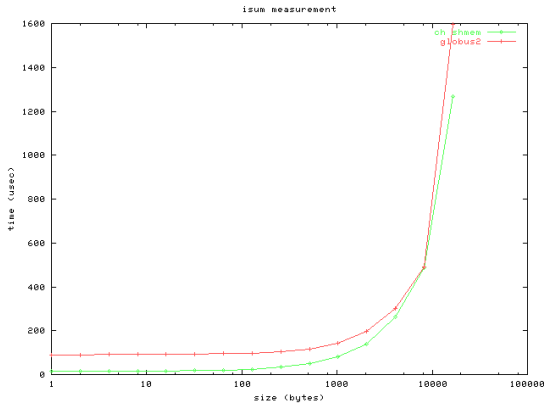


Figure 32: SMP level: Coll. operation: integer reduction

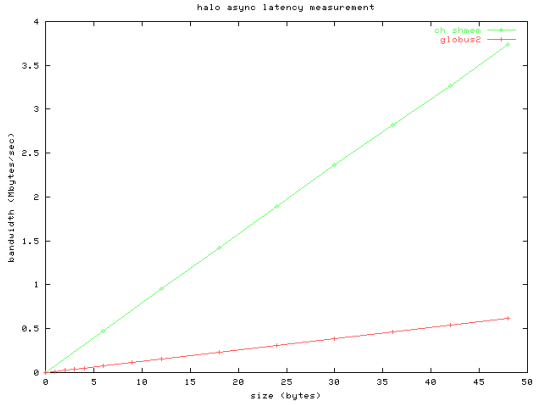


Figure 33: SMP level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 50 bytes

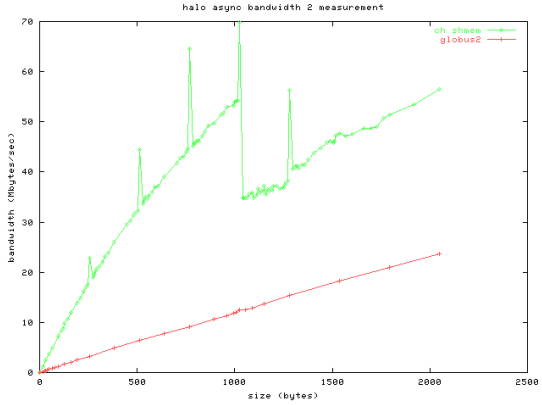


Figure 34: SMP level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 2 kbytes

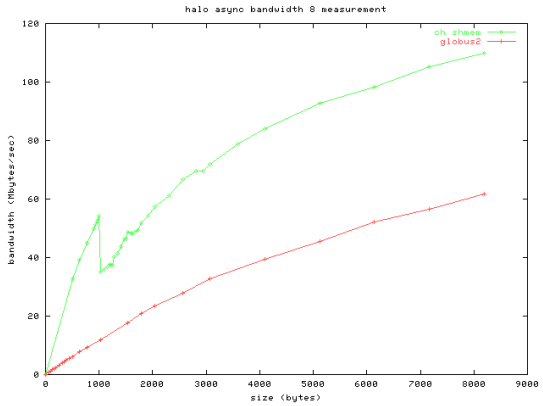


Figure 35: SMP level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 8 kbytes

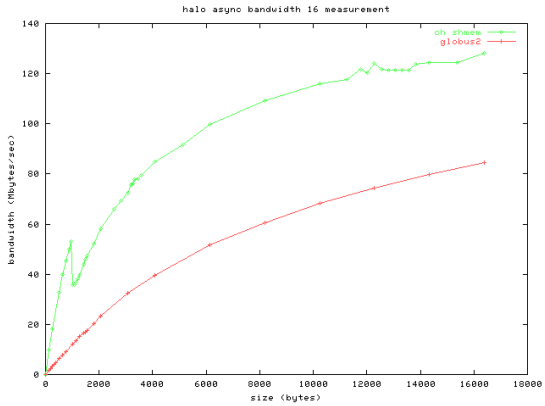


Figure 36: SMP level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 16 kbytes

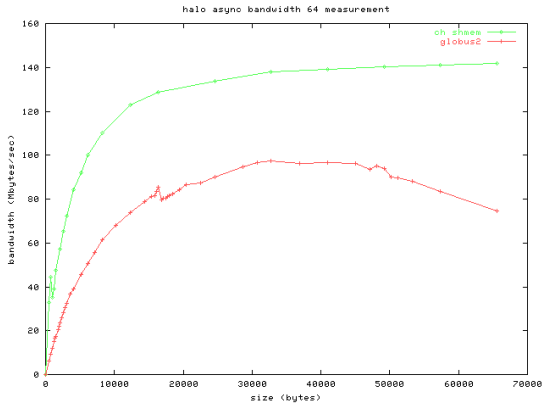


Figure 37: SMP level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 64 kbytes

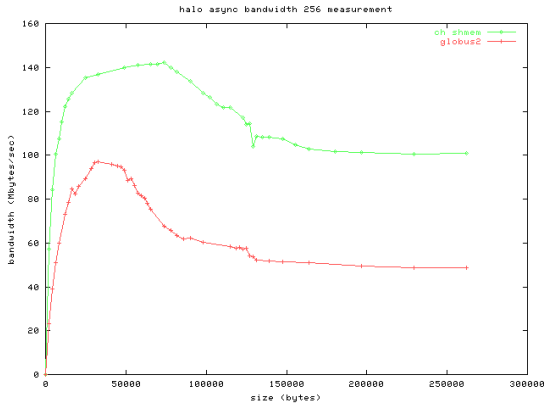


Figure 38: SMP level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 256 kbytes

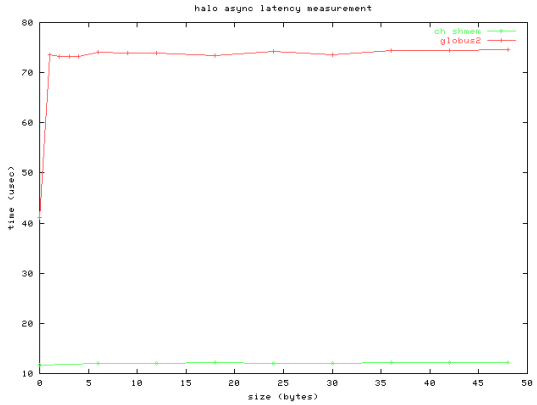


Figure 39: SMP level: Pt2pt (halo, non-blocking) timing, max. msg size 50 bytes

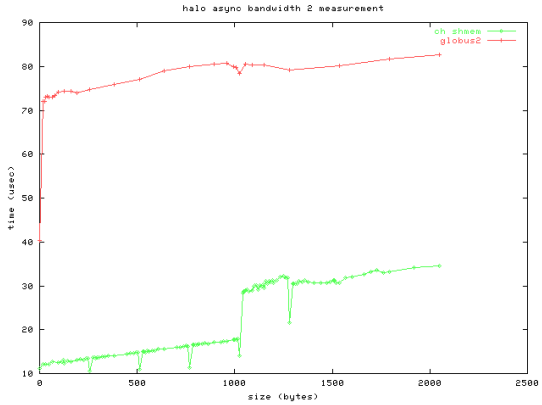


Figure 40: SMP level: Pt2pt (halo, non-blocking) timing, max. msg size 2 kbytes

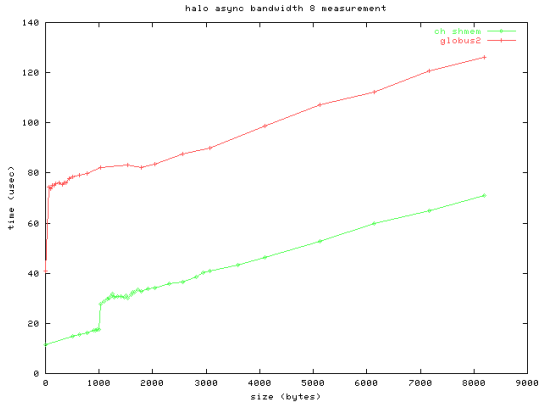


Figure 41: SMP level: Pt2pt (halo, non-blocking) timing, max. msg size 8 kbytes

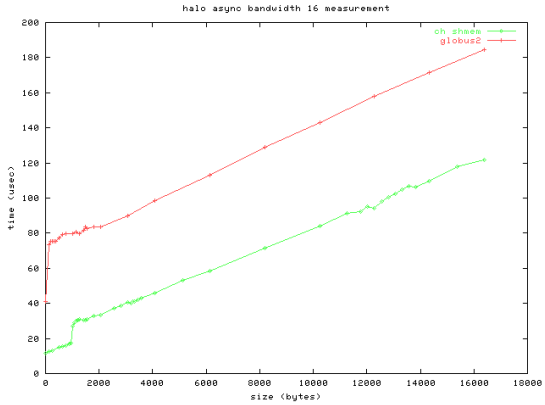


Figure 42: SMP level: Pt2pt (halo, non-blocking) timing, max. msg size 16 kbytes

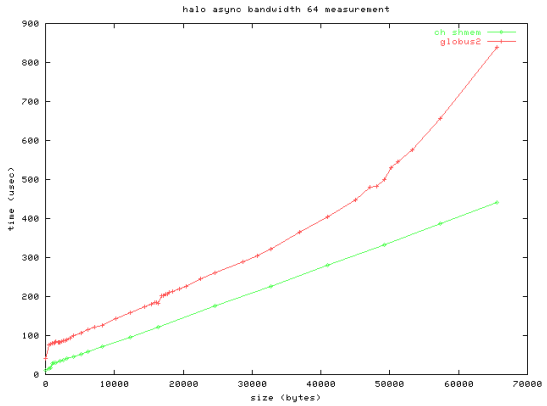


Figure 43: SMP level: Pt2pt (halo, non-blocking) timing, max. msg size 64 kbytes

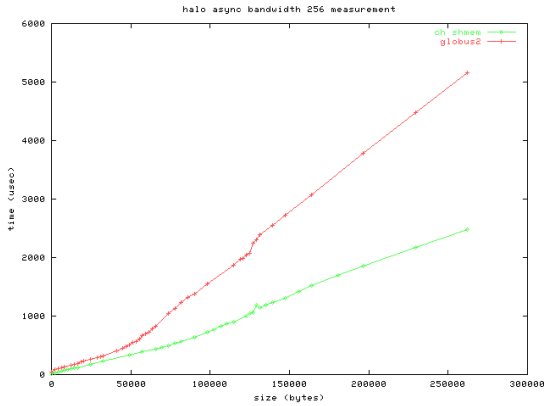


Figure 44: SMP level: Pt2pt (halo, non-blocking) timing, max. msg size 256 kbytes

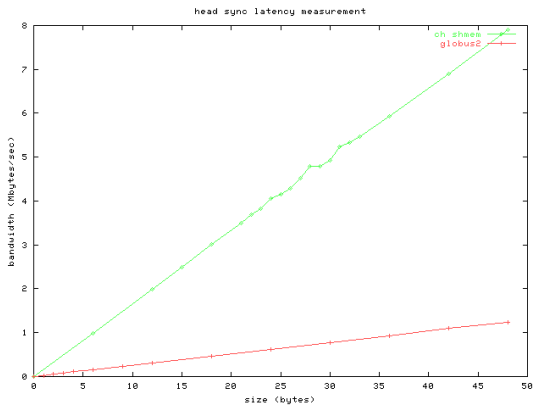


Figure 45: SMP level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 50 bytes

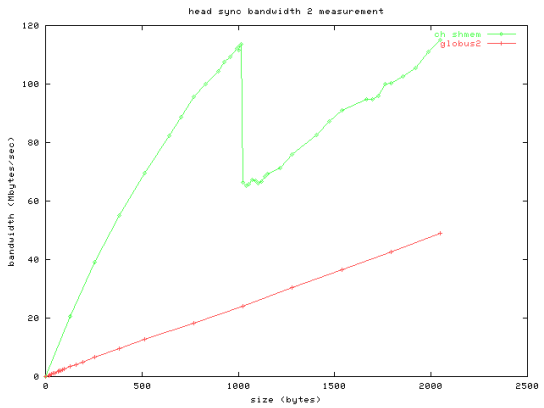


Figure 46: SMP level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 2 kbytes

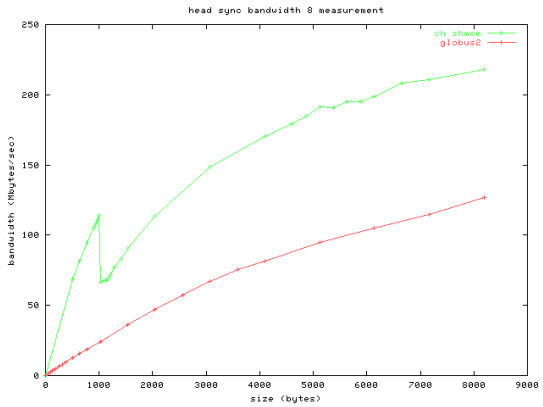


Figure 47: SMP level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 8 kbytes

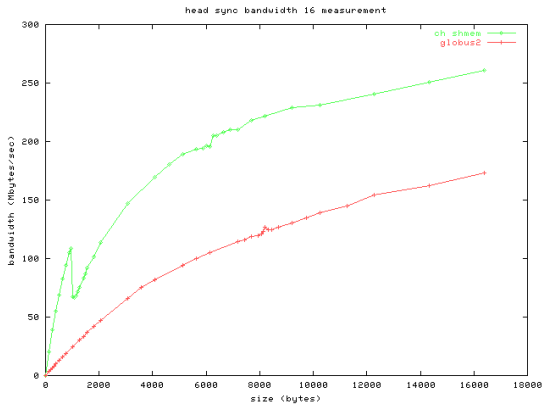


Figure 48: SMP level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 16 kbytes

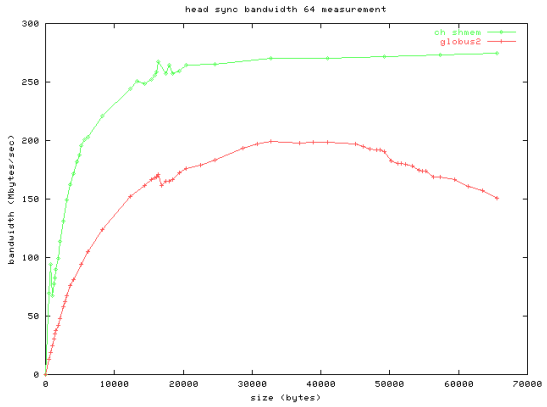


Figure 49: SMP level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 64 kbytes

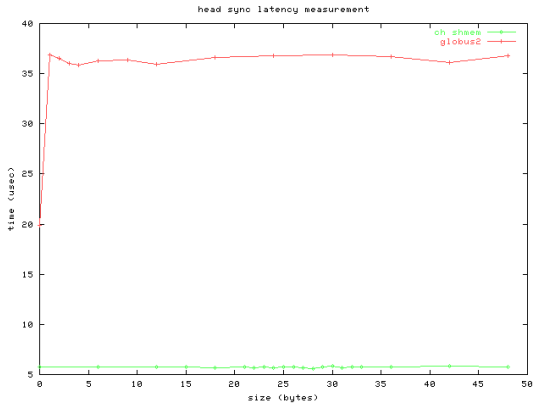


Figure 50: SMP level: Pt2pt (head-to-head, blocking) timing, max. msg size 50 bytes

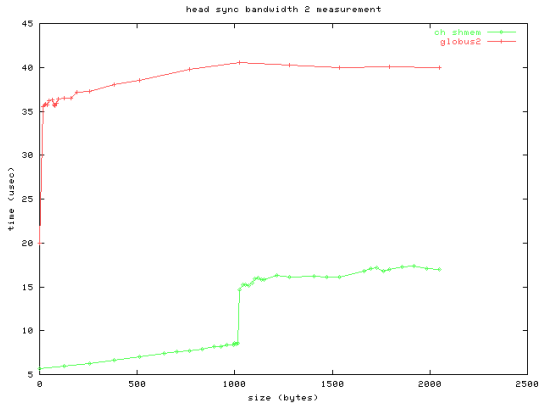


Figure 51: SMP level: Pt2pt (head-to-head, blocking) timing, max. msg size 2 kbytes

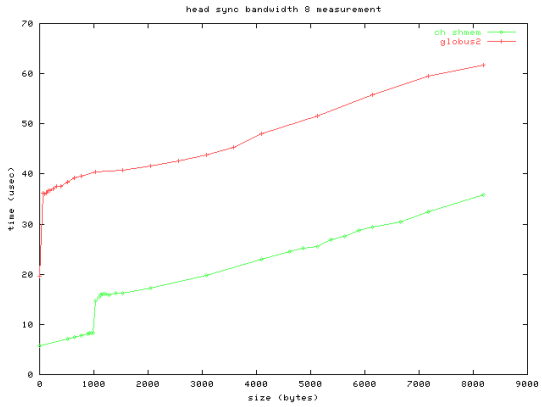


Figure 52: SMP level: Pt2pt (head-to-head, blocking) timing, max. msg size 8 kbytes

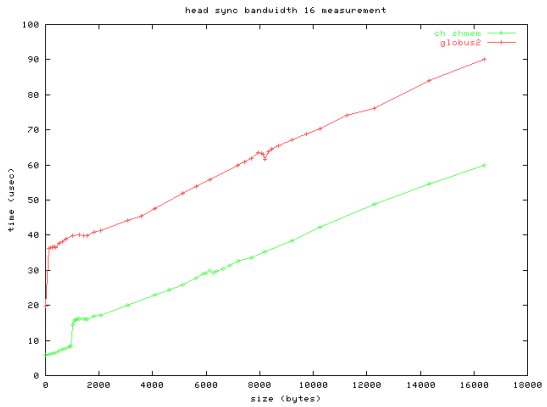


Figure 53: SMP level: Pt2pt (head-to-head, blocking) timing, max. msg size 16 kbytes

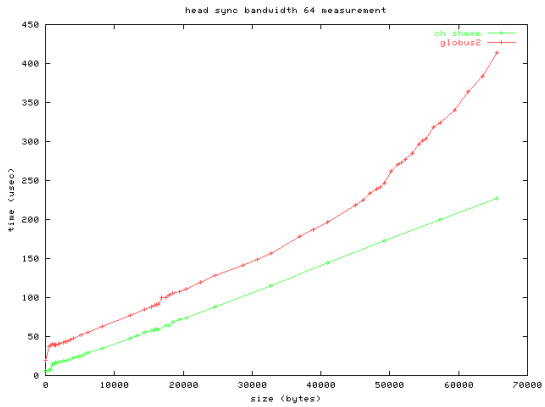


Figure 54: SMP level: Pt2pt (head-to-head, blocking) timing, max. msg size 64 kbytes

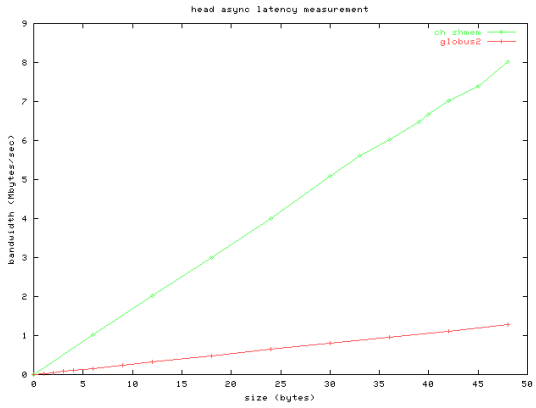


Figure 55: SMP level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 50 bytes

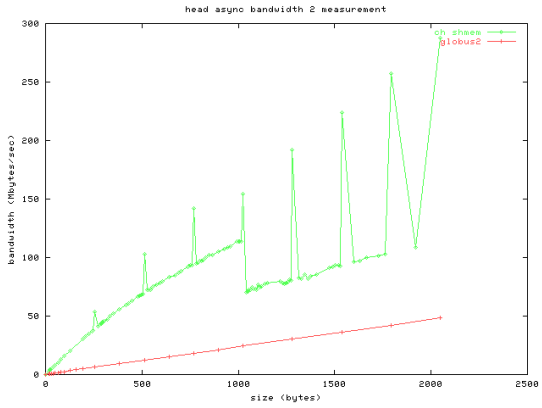


Figure 56: SMP level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 2 kbytes

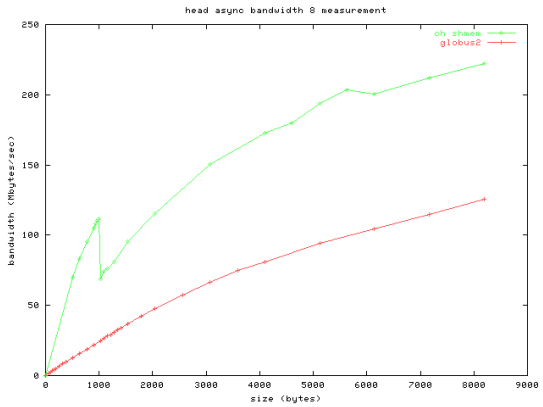


Figure 57: SMP level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 8 kbytes

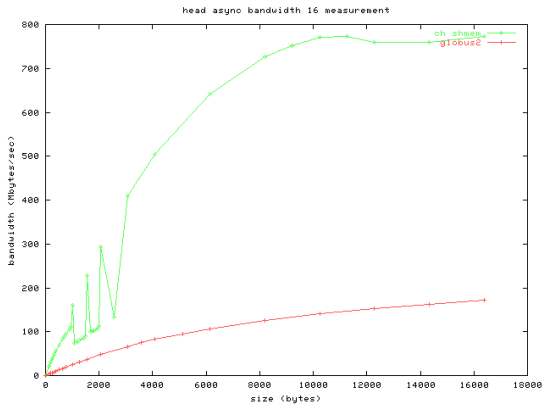


Figure 58: SMP level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 16 kbytes

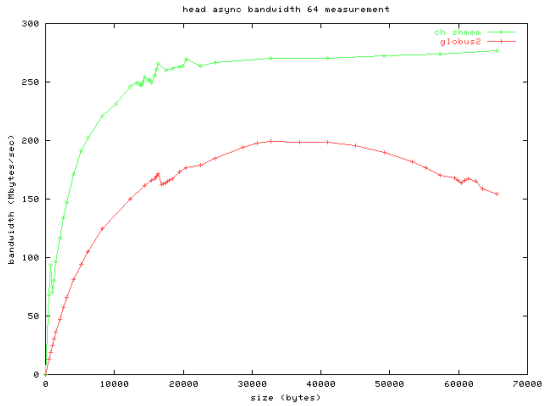


Figure 59: SMP level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 64 kbytes

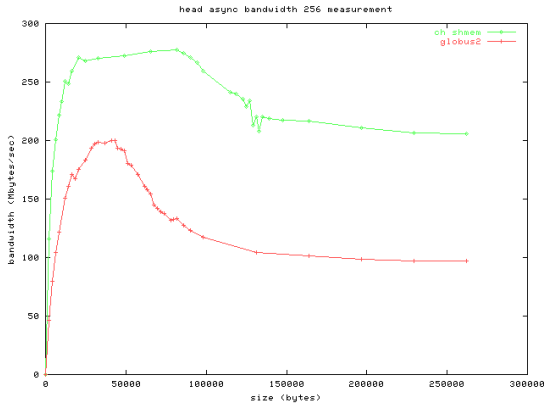


Figure 60: SMP level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 256 kbytes

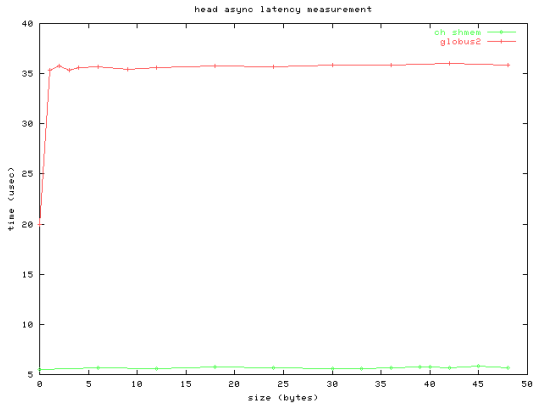


Figure 61: SMP level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 50 bytes

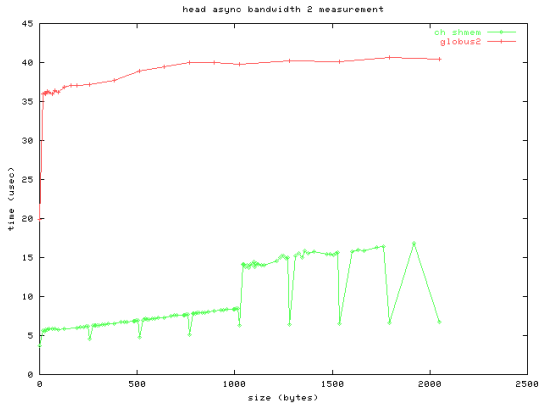


Figure 62: SMP level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 2 kbytes

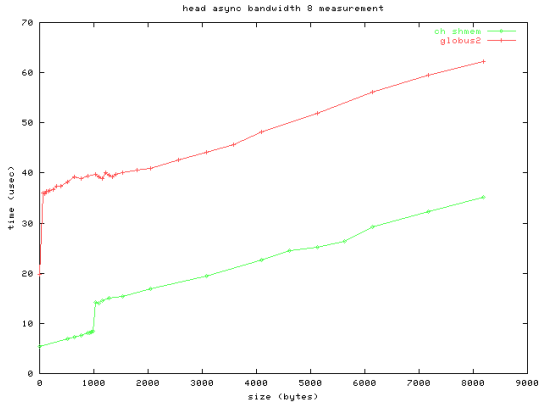


Figure 63: SMP level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 8 kbytes

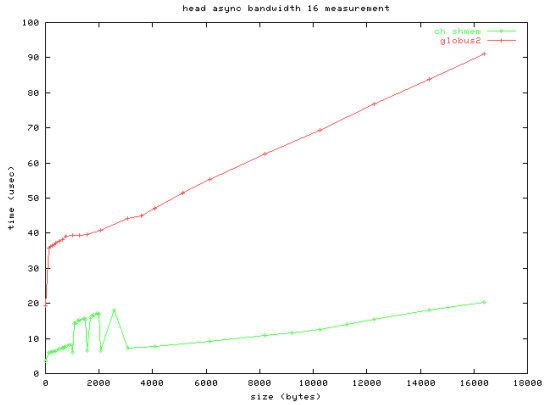


Figure 64: SMP level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 16 kbytes

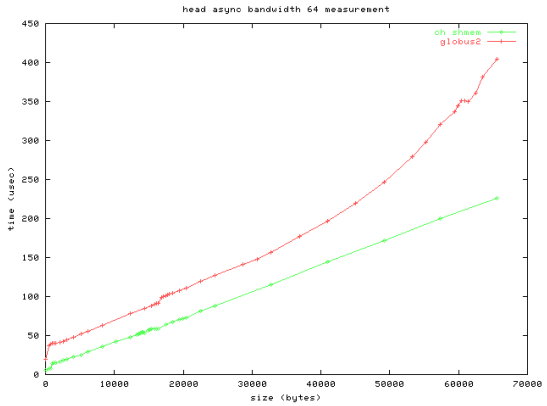


Figure 65: SMP level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 64 kbytes

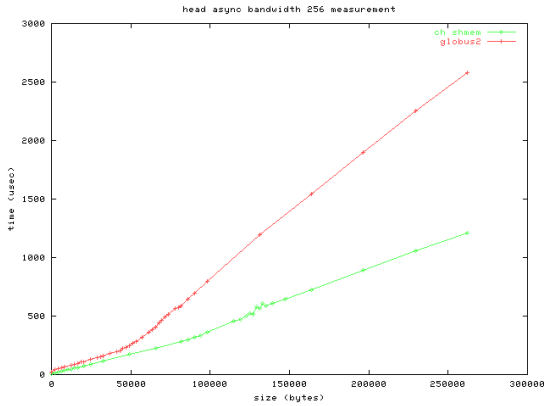


Figure 66: SMP level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 256 kbytes

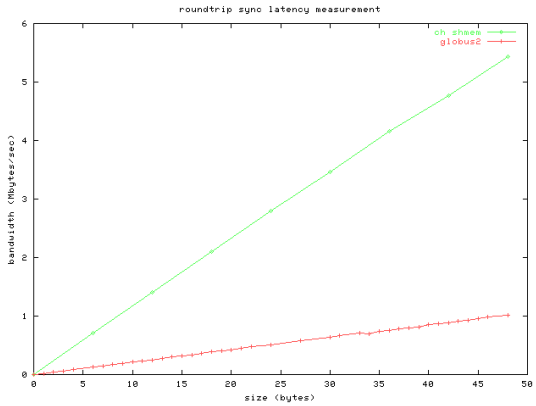


Figure 67: SMP level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 50 bytes

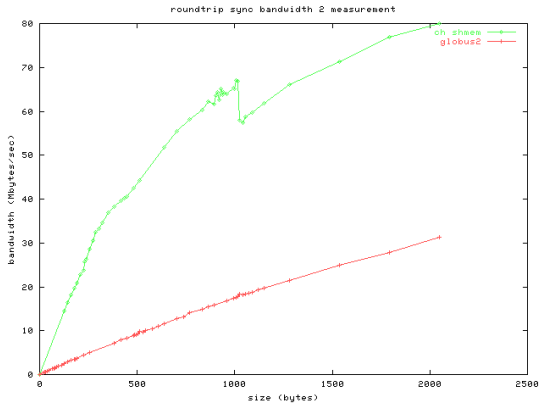


Figure 68: SMP level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 2 kbytes

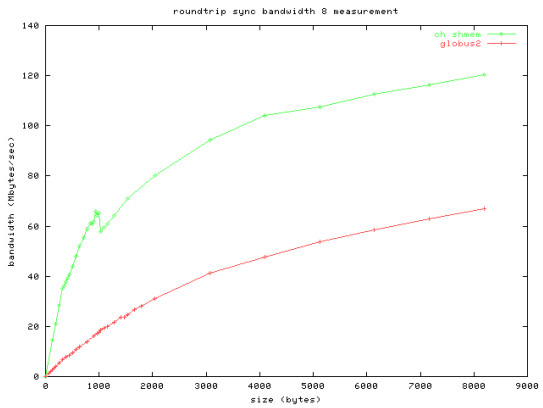


Figure 69: SMP level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 8 kbytes

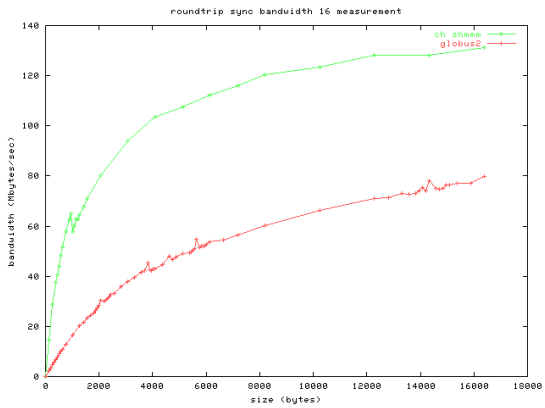


Figure 70: SMP level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 16 kbytes

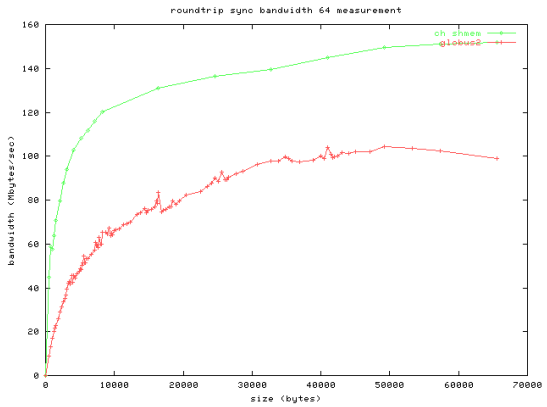


Figure 71: SMP level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 64 kbytes

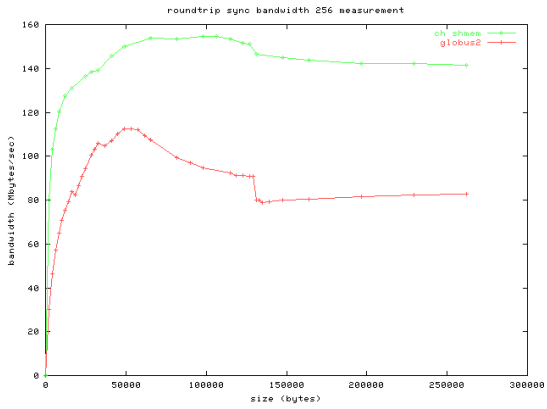


Figure 72: SMP level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 256 kbytes

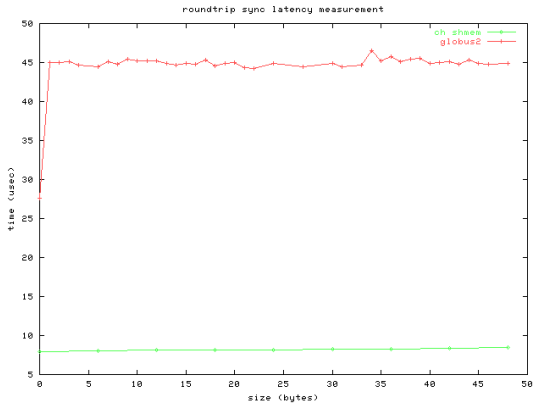


Figure 73: SMP level: Pt2pt (roundtrip, blocking) timing, max. msg size 50 bytes

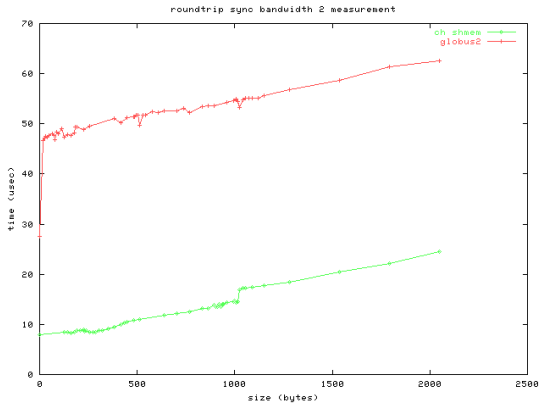


Figure 74: SMP level: Pt2pt (roundtrip, blocking) timing, max. msg size 2 kbytes

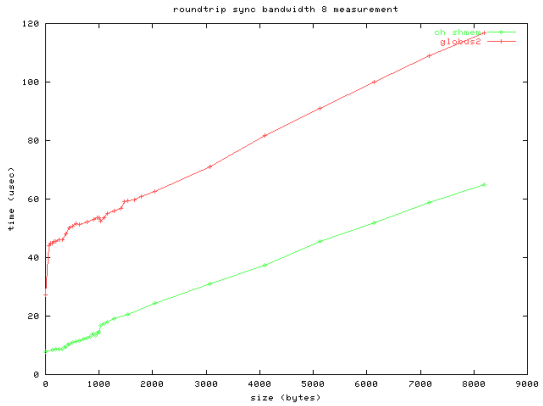


Figure 75: SMP level: Pt2pt (roundtrip, blocking) timing, max. msg size 8 kbytes

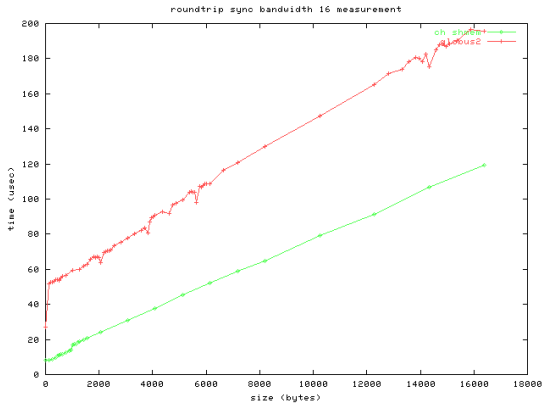


Figure 76: SMP level: Pt2pt (roundtrip, blocking) timing, max. msg size 16 kbytes

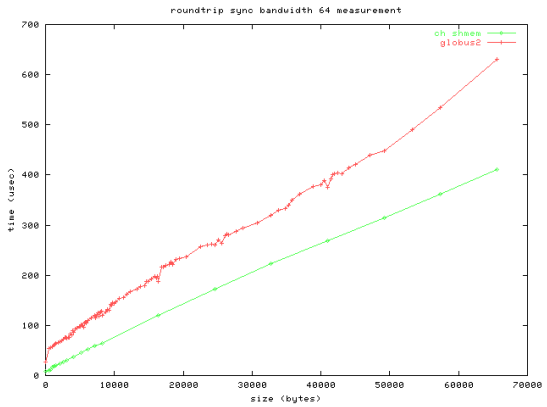


Figure 77: SMP level: Pt2pt (roundtrip, blocking) timing, max. msg size 64 kbytes

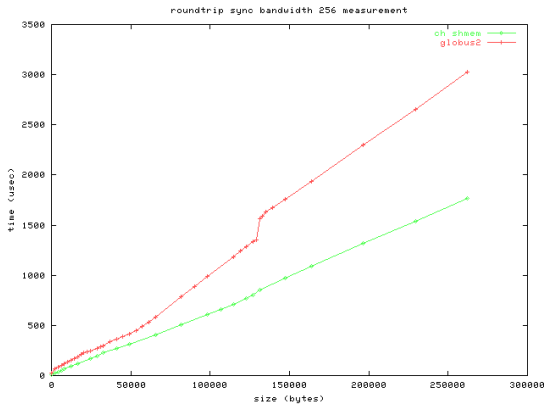


Figure 78: SMP level: Pt2pt (roundtrip, blocking) timing, max. msg size 256 kbytes

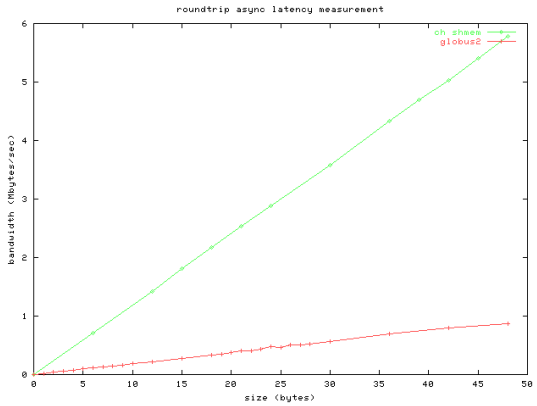


Figure 79: SMP level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 50 bytes

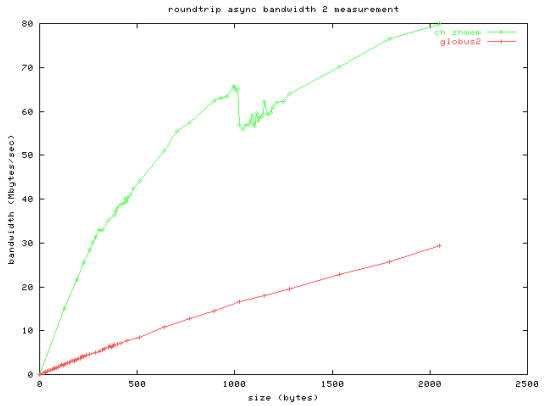


Figure 80: SMP level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 2 kbytes

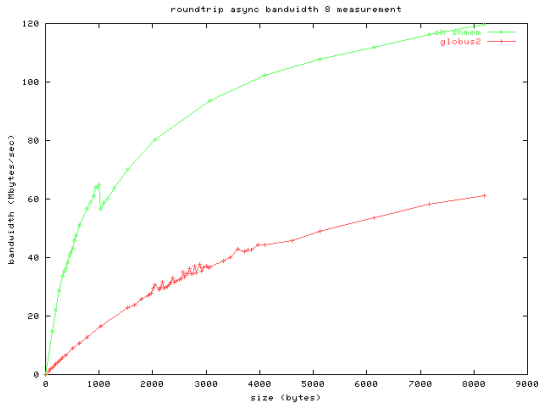


Figure 81: SMP level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 8 kbytes

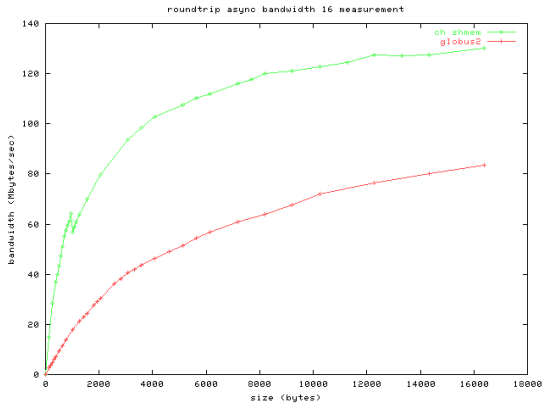


Figure 82: SMP level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 16 kbytes

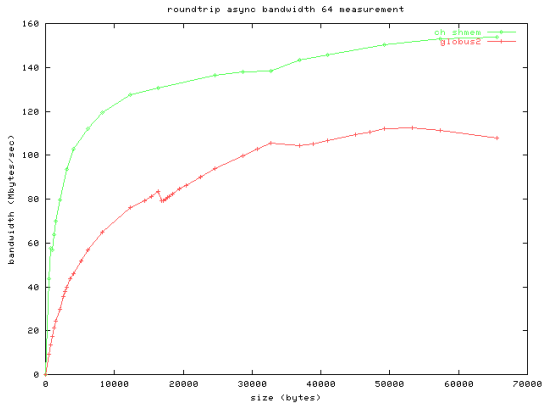


Figure 83: SMP level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 64 kbytes

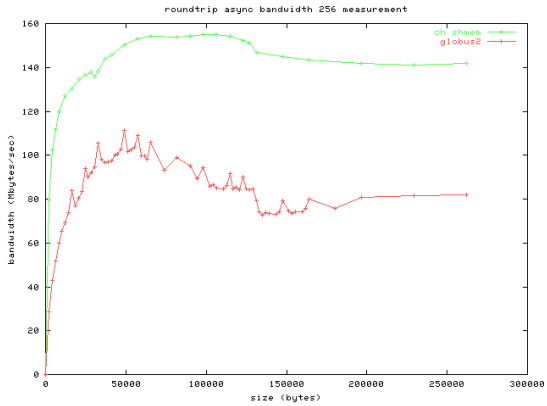


Figure 84: SMP level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 256 kbytes

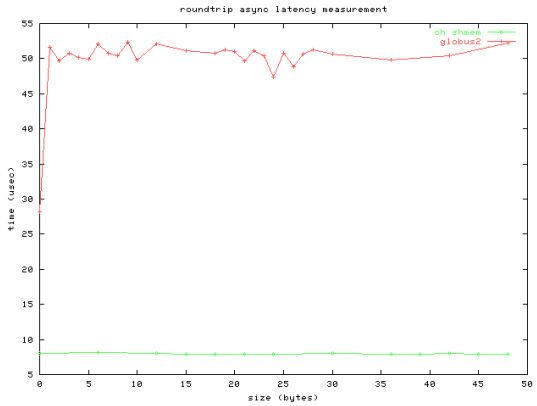


Figure 85: SMP level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 50 bytes

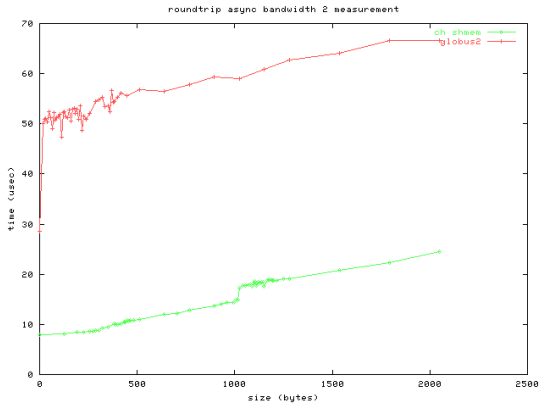


Figure 86: SMP level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 2 kbytes

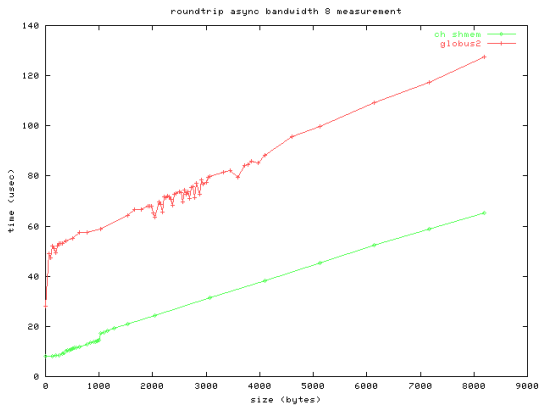


Figure 87: SMP level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 8 kbytes

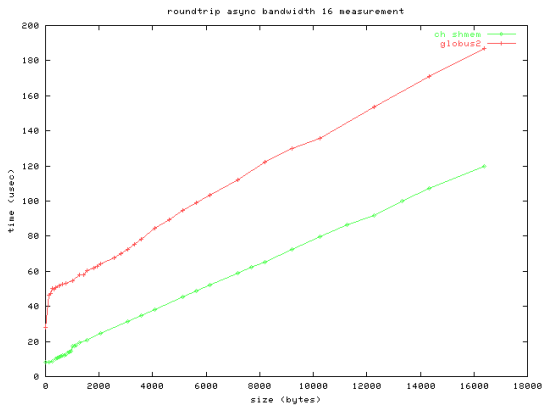


Figure 88: SMP level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 16 kbytes

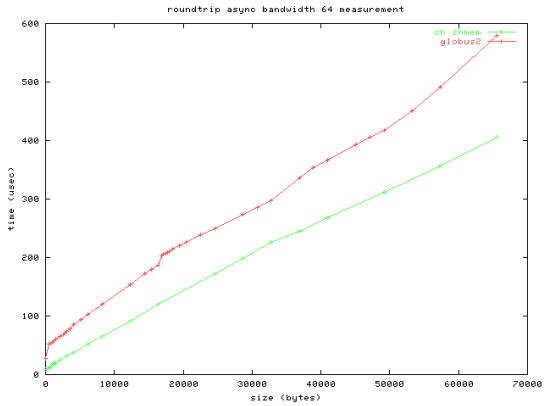


Figure 89: SMP level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 64 kbytes

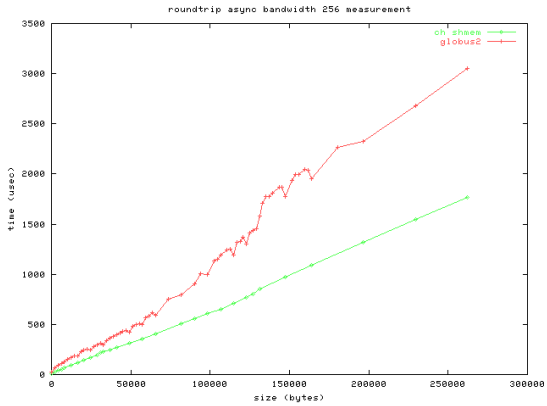


Figure 90: SMP level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 256 kbytes

B.2 Point-to-Point Tests on Cluster Level with two Processes

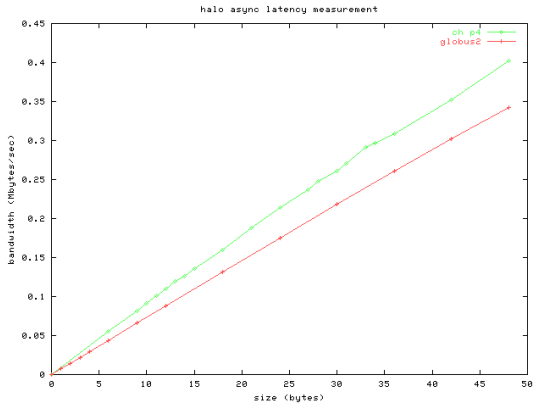


Figure 91: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 50 bytes

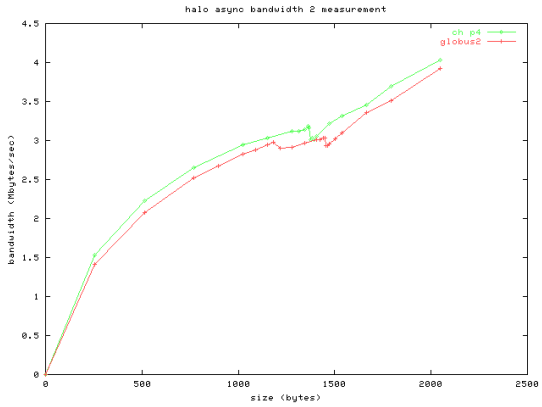


Figure 92: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 2 kbytes

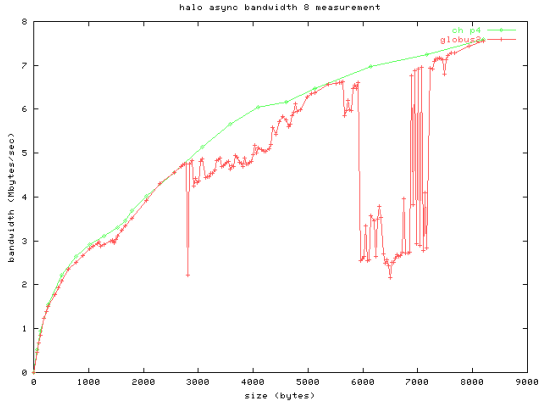


Figure 93: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 8 kbytes

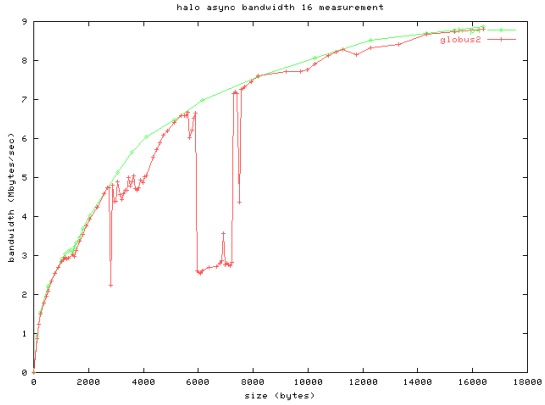


Figure 94: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 16 kbytes

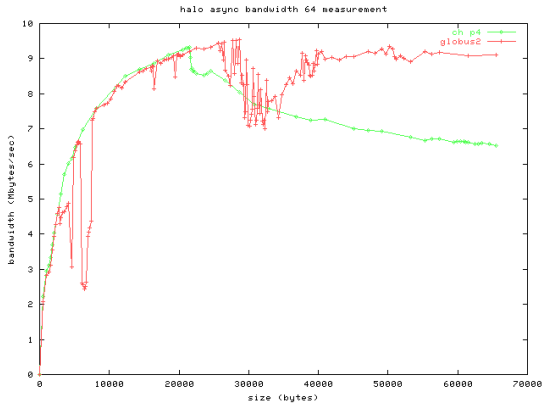


Figure 95: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 64 kbytes

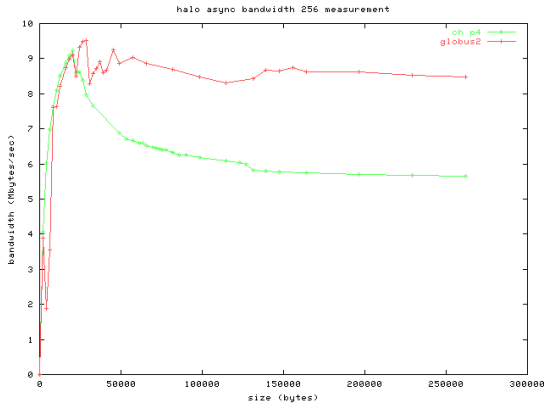


Figure 96: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 256 kbytes

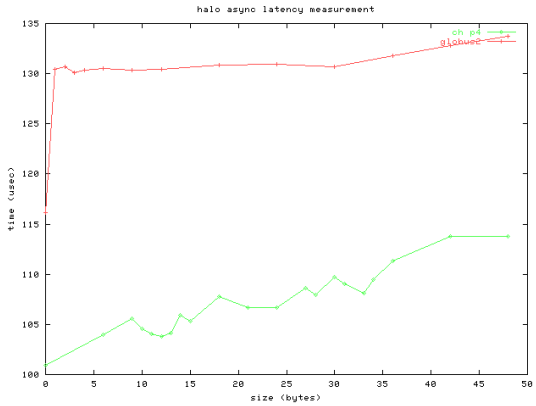


Figure 97: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 50 bytes

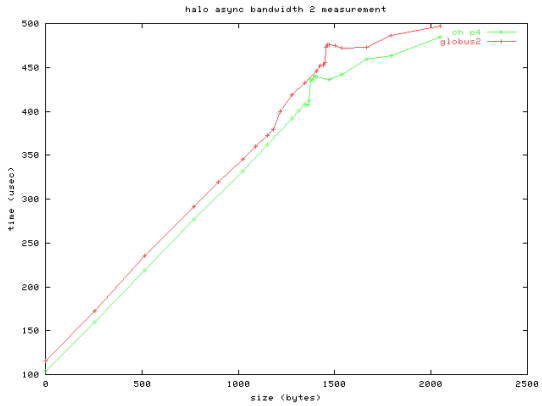


Figure 98: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 2 kbytes

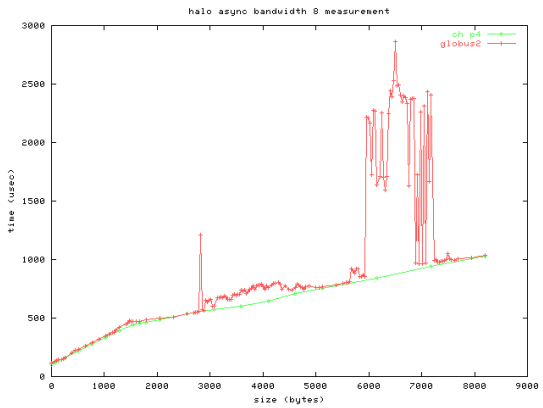


Figure 99: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 8 kbytes

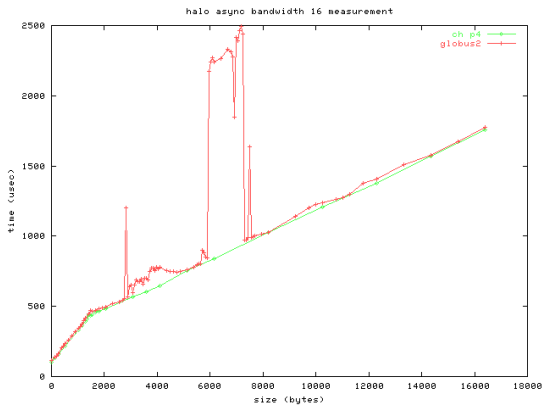


Figure 100: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 16 kbytes

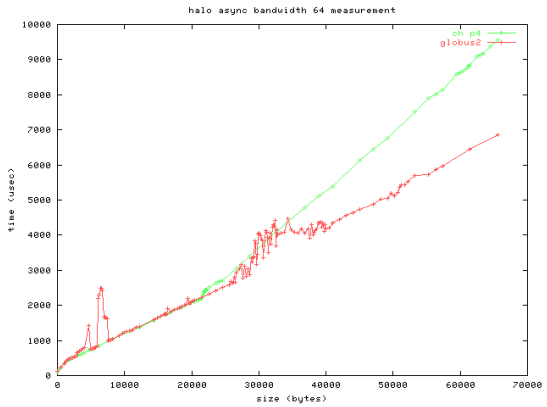


Figure 101: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 64 kbytes

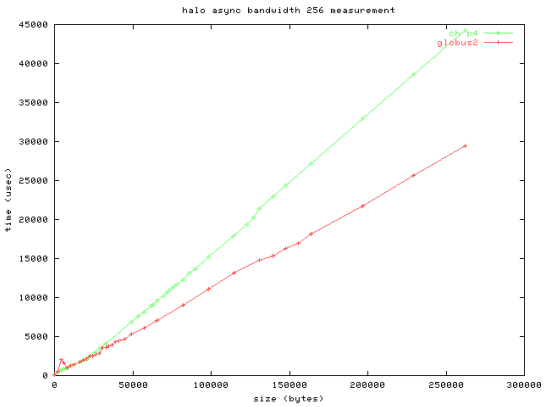


Figure 102: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 256 kbytes

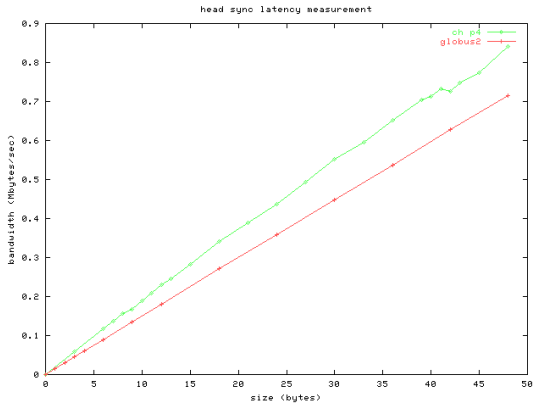


Figure 103: LAN level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 50 bytes

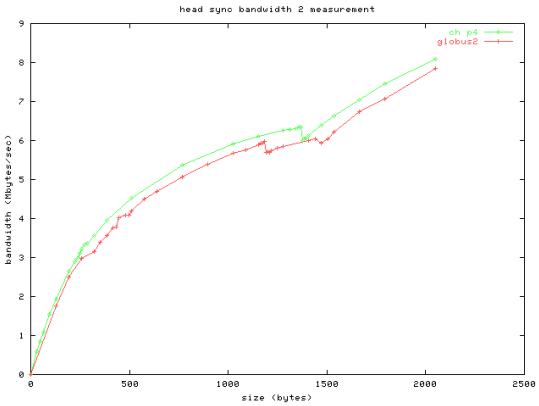


Figure 104: LAN level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 2 kbytes

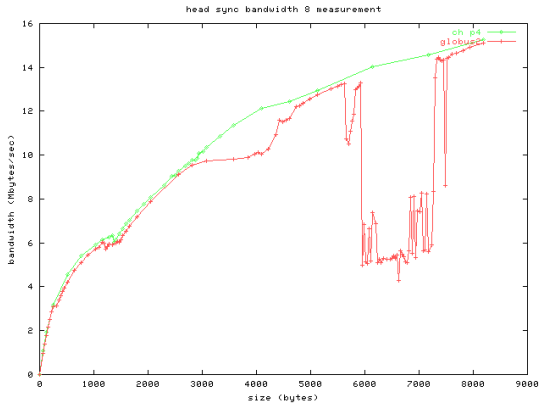


Figure 105: LAN level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 8 kbytes

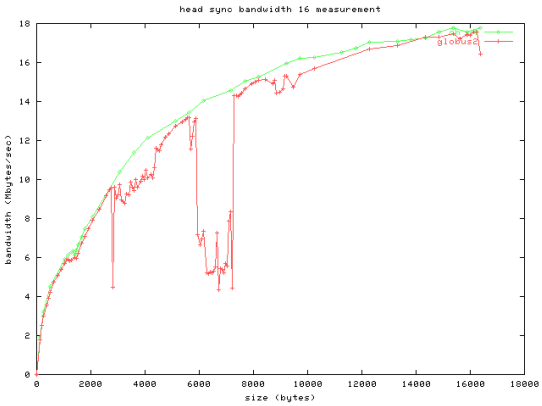


Figure 106: LAN level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 16 kbytes

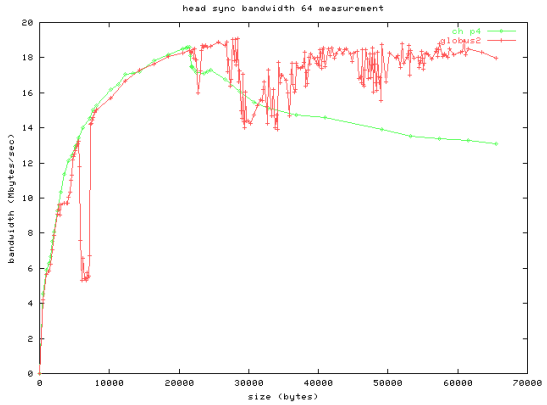


Figure 107: LAN level: Pt2pt (head-to-head, blocking) bandwidth, max. msg size 64 kbytes

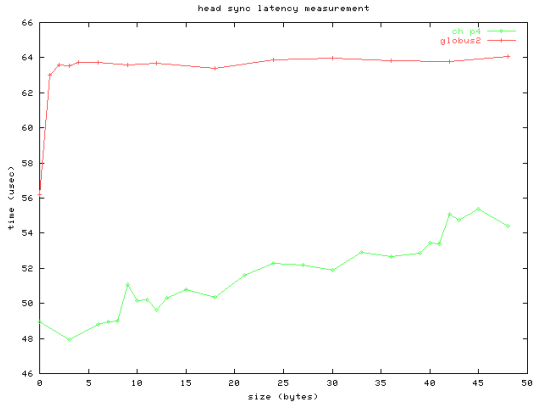


Figure 108: LAN level: Pt2pt (head-to-head, blocking) timing, max. msg size 50 bytes

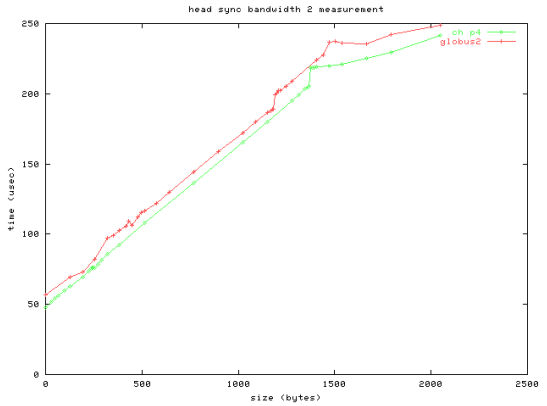


Figure 109: LAN level: Pt2pt (head-to-head, blocking) timing, max. msg size 2 kbytes

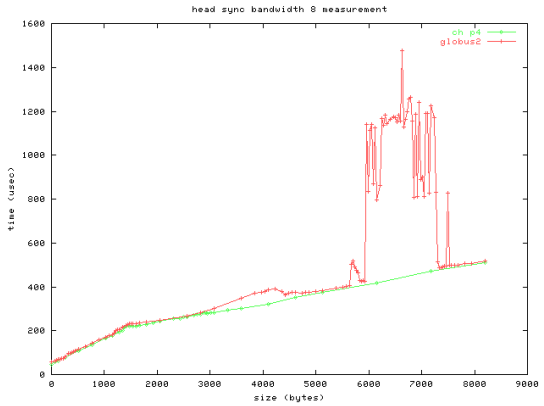


Figure 110: LAN level: Pt2pt (head-to-head, blocking) timing, max. msg size 8 kbytes

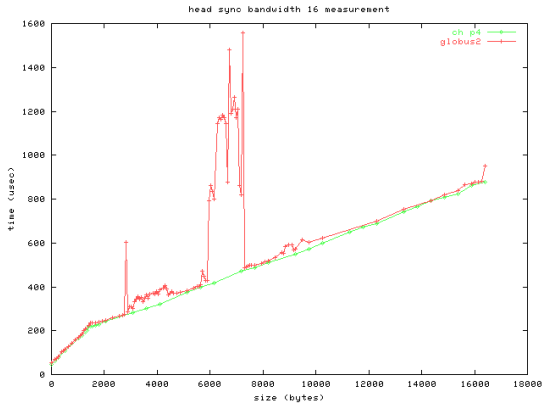


Figure 111: LAN level: Pt2pt (head-to-head, blocking) timing, max. msg size 16 kbytes

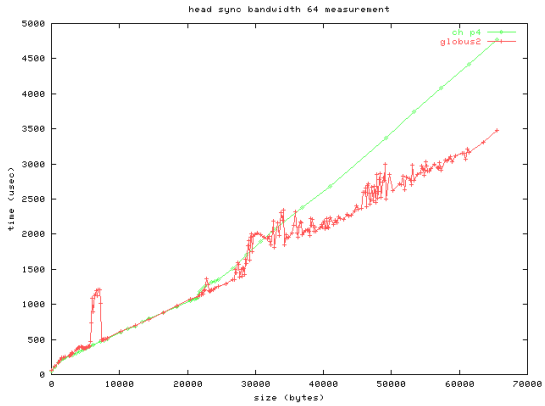


Figure 112: LAN level: Pt2pt (head-to-head, blocking) timing, max. msg size 64 kbytes

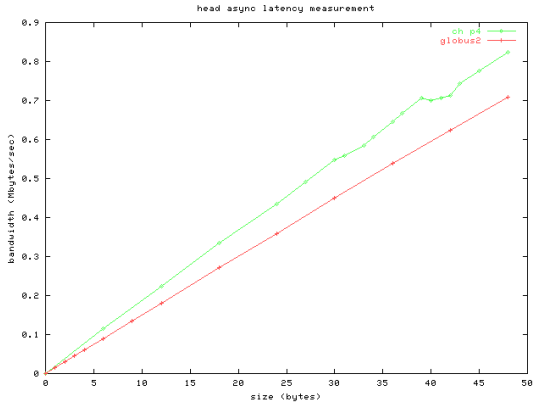


Figure 113: LAN level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 50 bytes

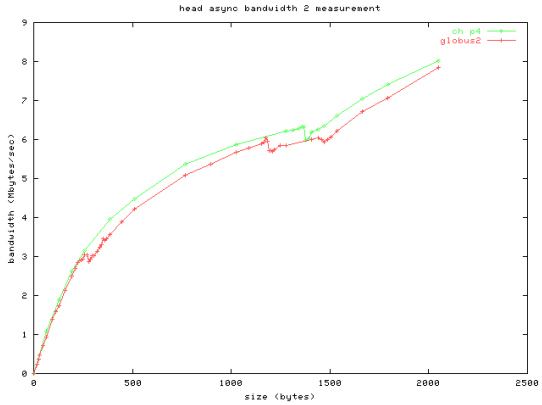


Figure 114: LAN level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 2 kbytes

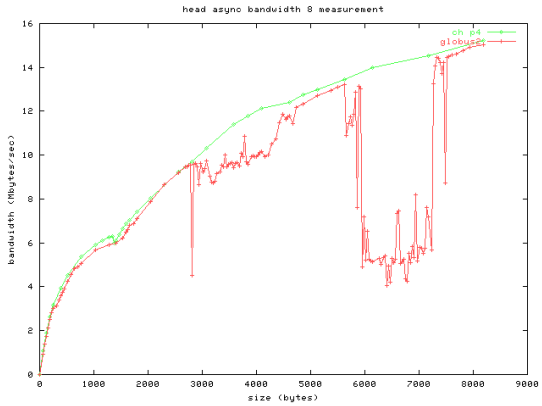


Figure 115: LAN level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 8 kbytes

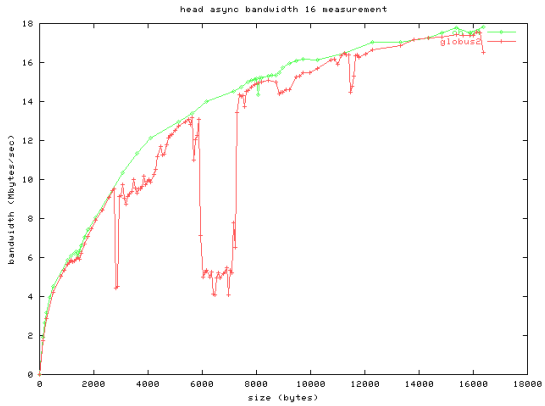


Figure 116: LAN level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 16 kbytes

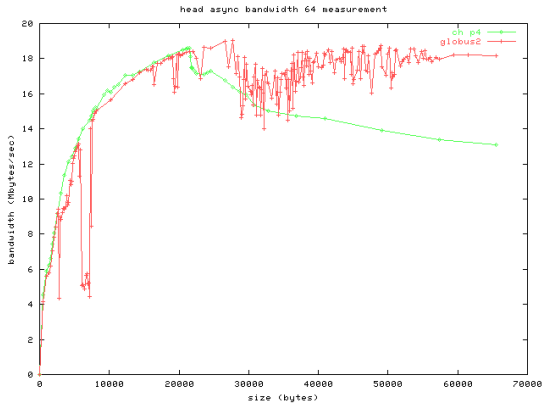


Figure 117: LAN level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 64 kbytes

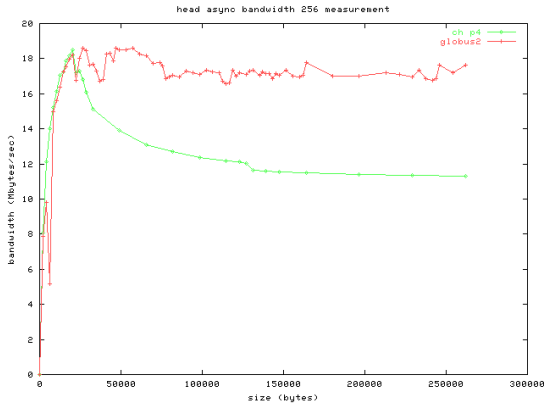


Figure 118: LAN level: Pt2pt (head-to-head, non-blocking) bandwidth, max. msg size 256 kbytes

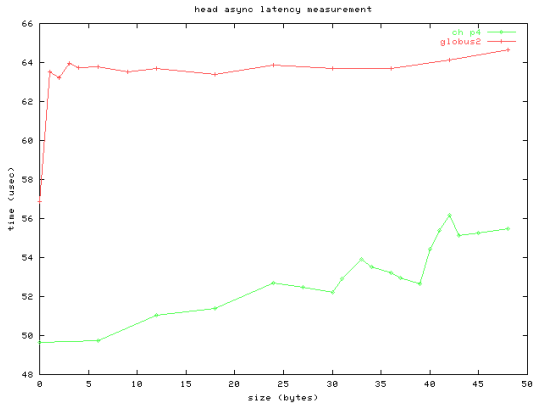


Figure 119: LAN level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 50 bytes

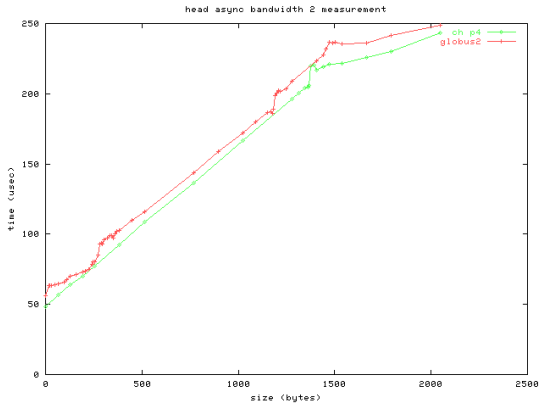


Figure 120: LAN level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 2 kbytes

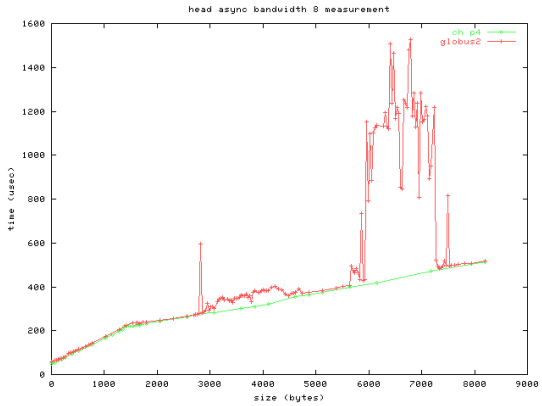


Figure 121: LAN level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 8 kbytes

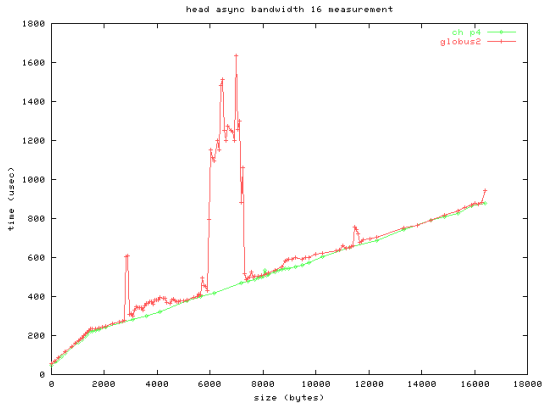


Figure 122: LAN level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 16 kbytes

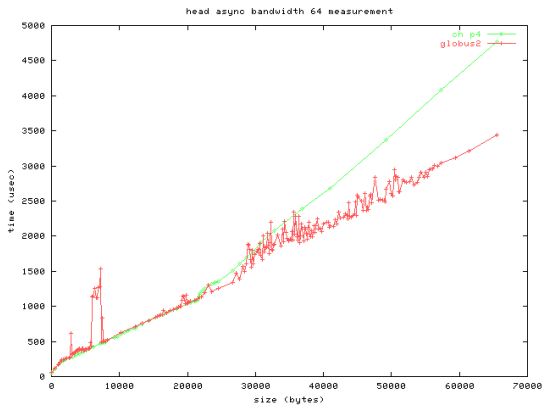


Figure 123: LAN level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 64 kbytes

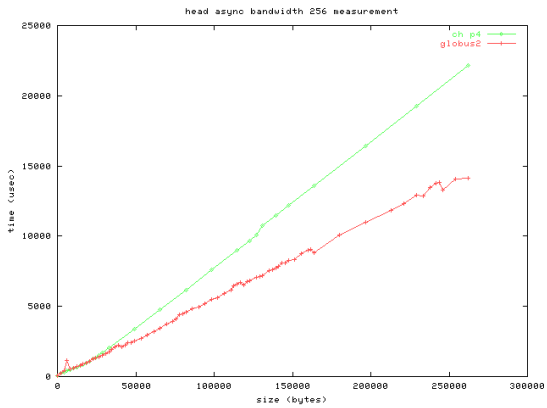


Figure 124: LAN level: Pt2pt (head-to-head, non-blocking) timing, max. msg size 256 kbytes

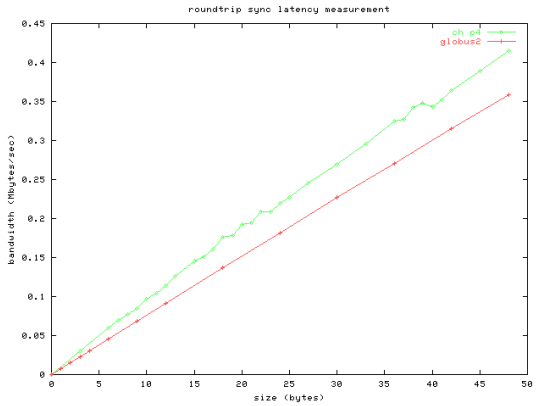


Figure 125: LAN level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 50 bytes

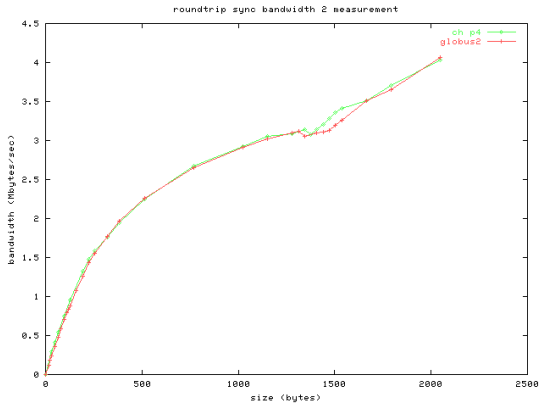


Figure 126: LAN level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 2 kbytes

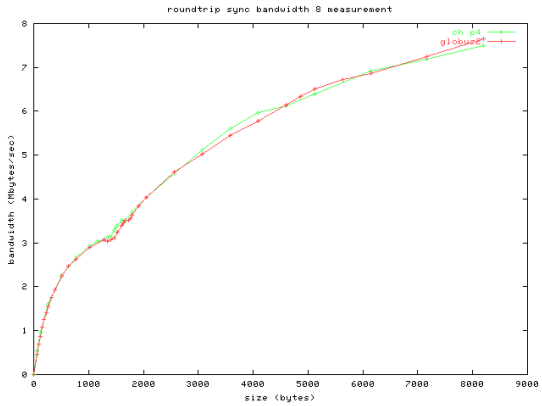


Figure 127: LAN level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 8 kbytes

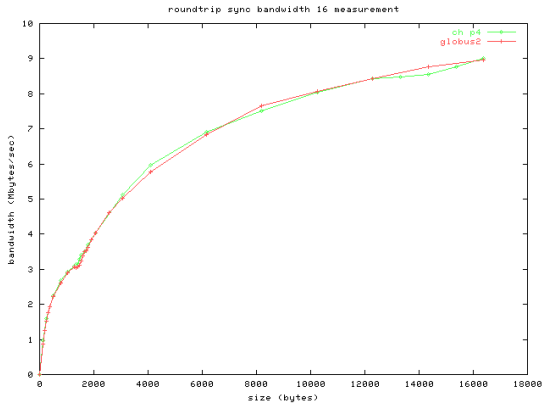


Figure 128: LAN level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 16 kbytes

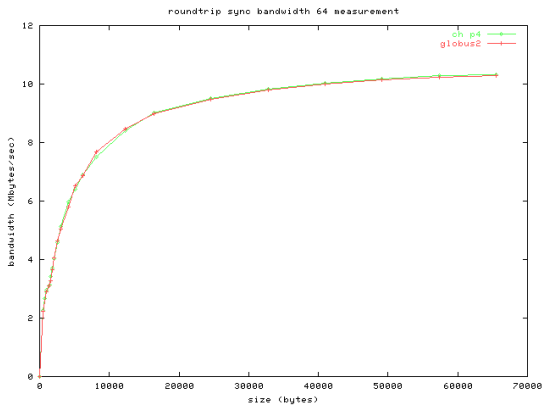


Figure 129: LAN level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 64 kbytes

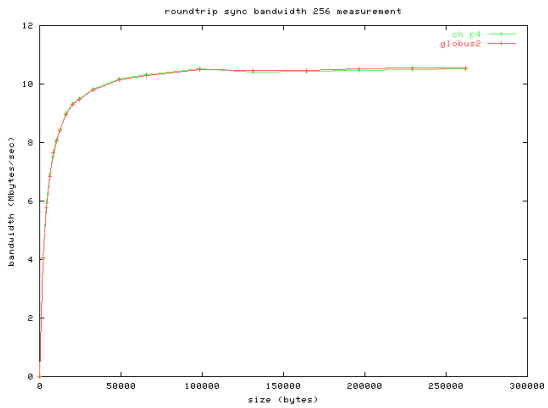


Figure 130: LAN level: Pt2pt (roundtrip, blocking) bandwidth, max. msg size 256 kbytes

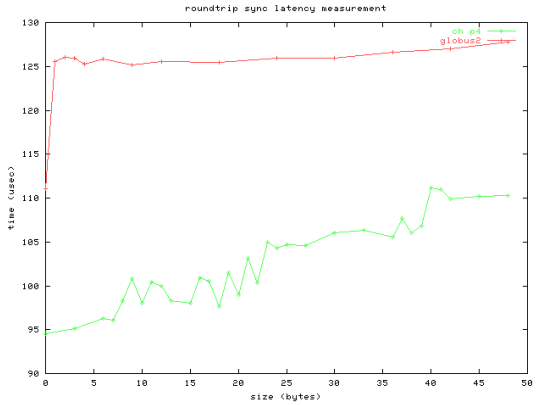


Figure 131: LAN level: Pt2pt (roundtrip, blocking) timing, max. msg size 50 bytes

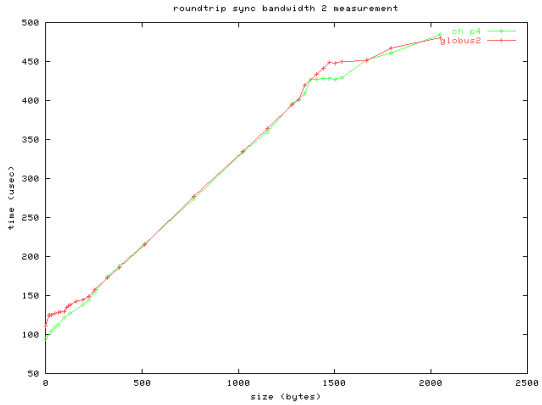


Figure 132: LAN level: Pt2pt (roundtrip, blocking) timing, max. msg size 2 kbytes

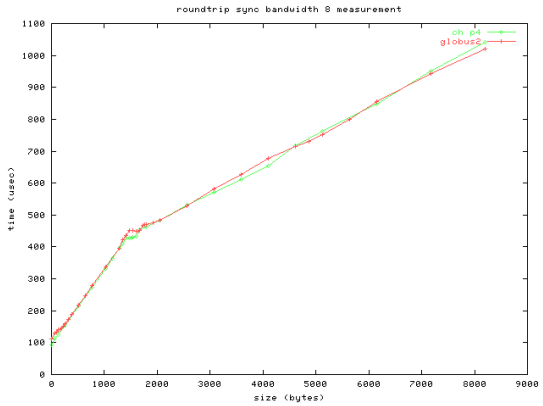


Figure 133: LAN level: Pt2pt (roundtrip, blocking) timing, max. msg size 8 kbytes

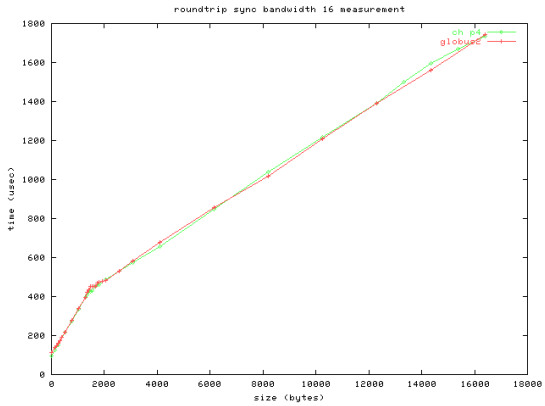


Figure 134: LAN level: Pt2pt (roundtrip, blocking) timing, max. msg size 16 kbytes

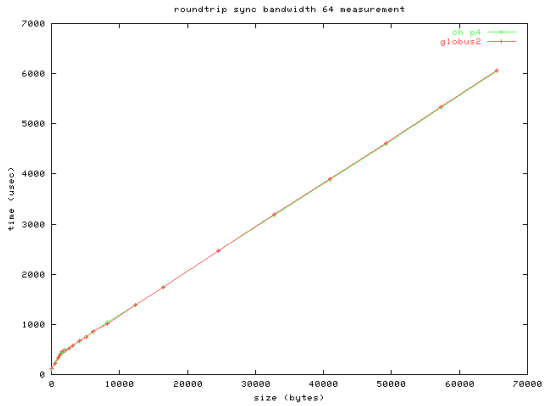


Figure 135: LAN level: Pt2pt (roundtrip, blocking) timing, max. msg size 64 kbytes

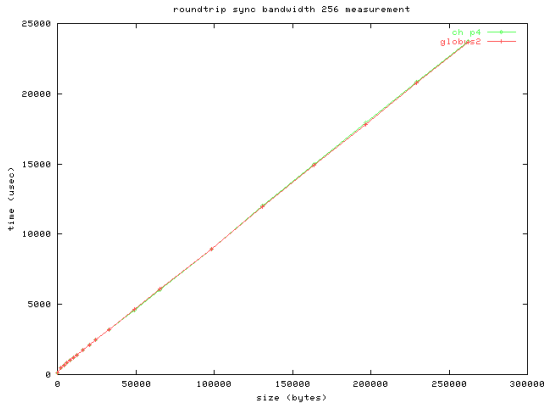


Figure 136: LAN level: Pt2pt (roundtrip, blocking) timing, max. msg size 256 kbytes

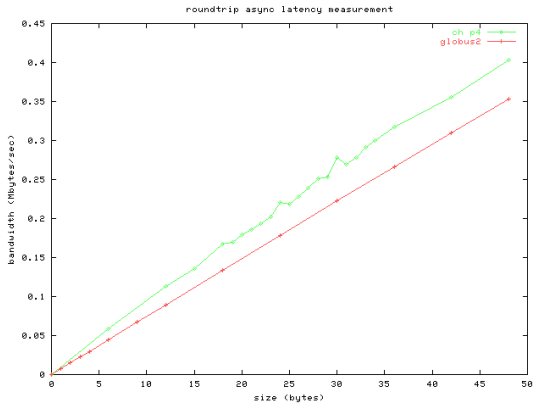


Figure 137: LAN level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 50 bytes

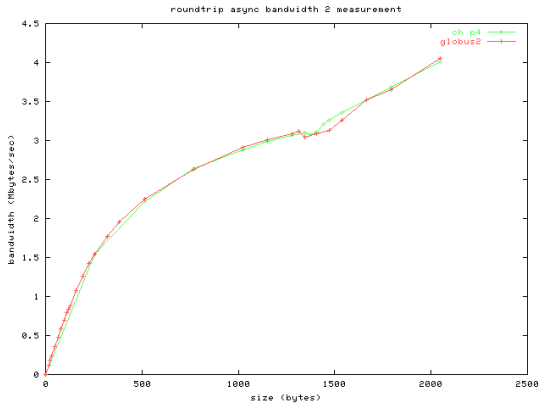


Figure 138: LAN level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 2 kbytes

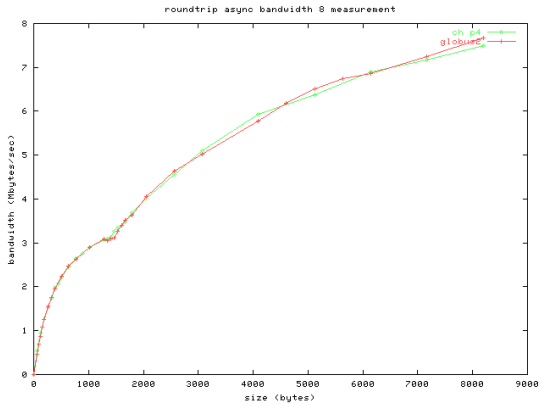


Figure 139: LAN level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 8 kbytes

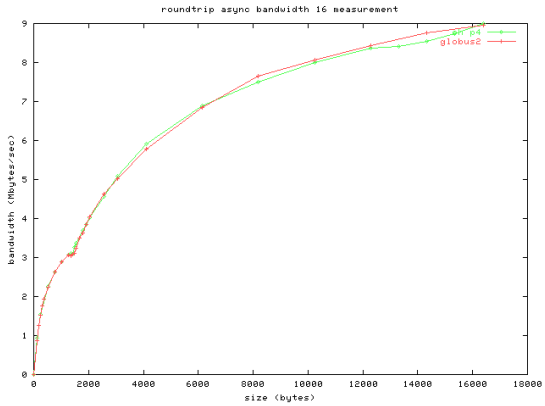


Figure 140: LAN level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 16 kbytes

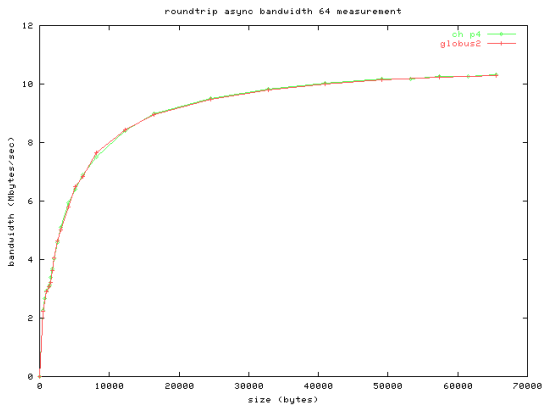


Figure 141: LAN level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 64 kbytes

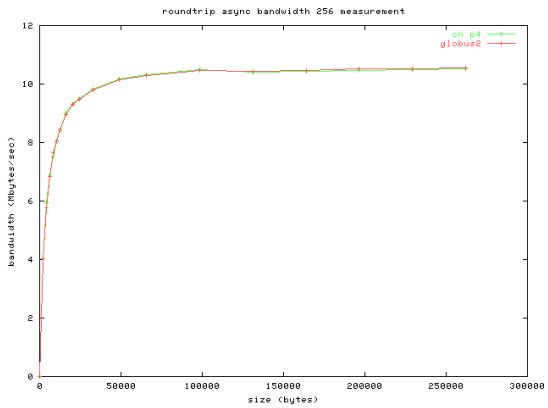


Figure 142: LAN level: Pt2pt (roundtrip, non-blocking) bandwidth, max. msg size 256 kbytes

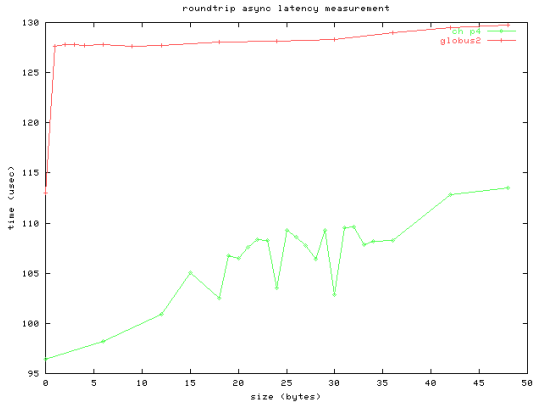


Figure 143: LAN level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 50 bytes

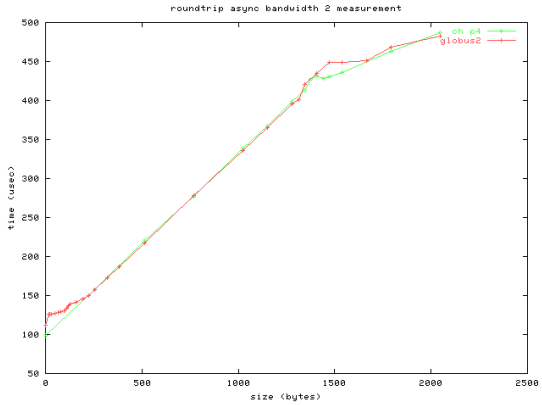


Figure 144: LAN level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 2 kbytes

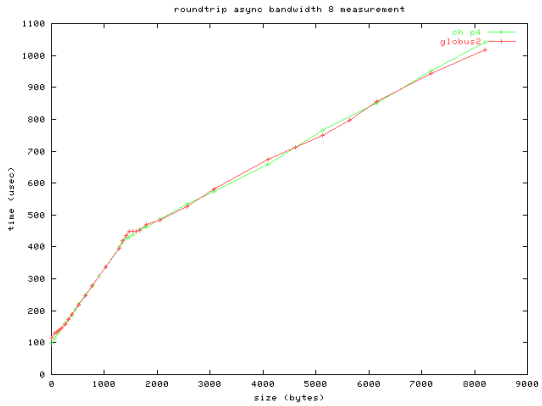


Figure 145: LAN level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 8 kbytes

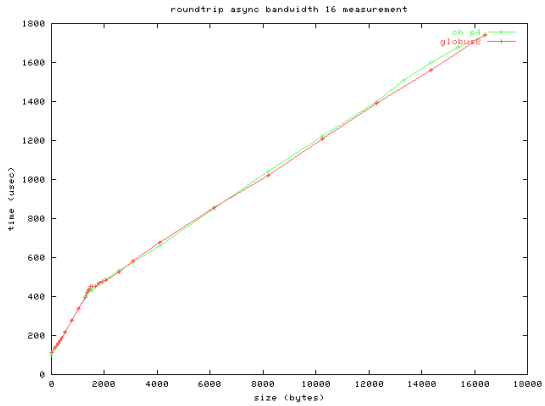


Figure 146: LAN level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 16 kbytes

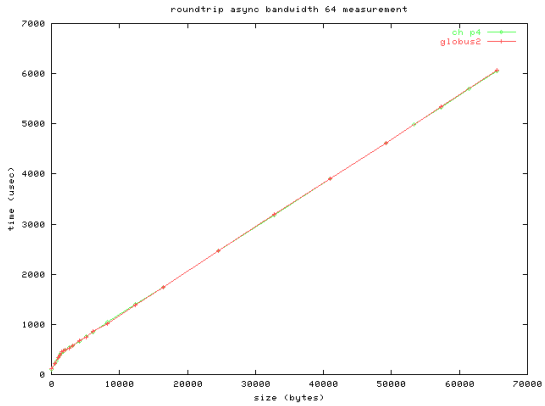


Figure 147: LAN level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 64 kbytes

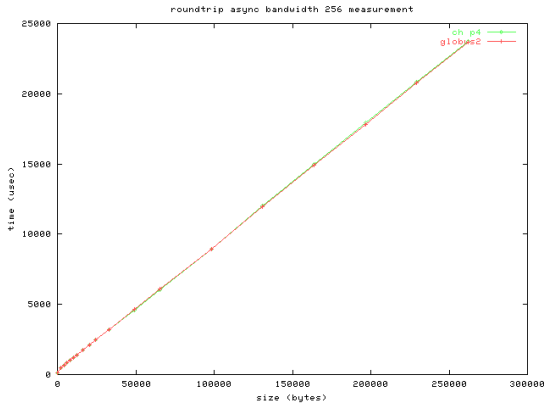


Figure 148: LAN level: Pt2pt (roundtrip, non-blocking) timing, max. msg size 256 kbytes

B.3 Point-to-Point Tests on Cluster Level with four Processes

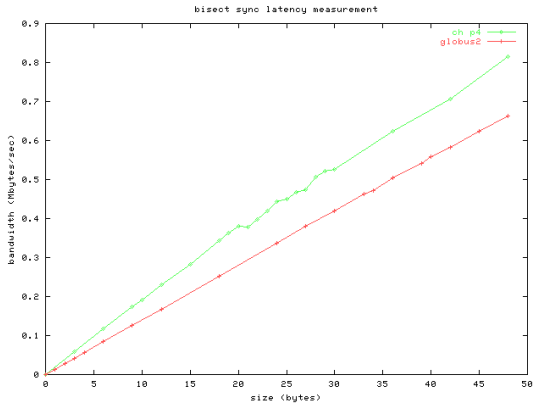


Figure 149: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 50 bytes

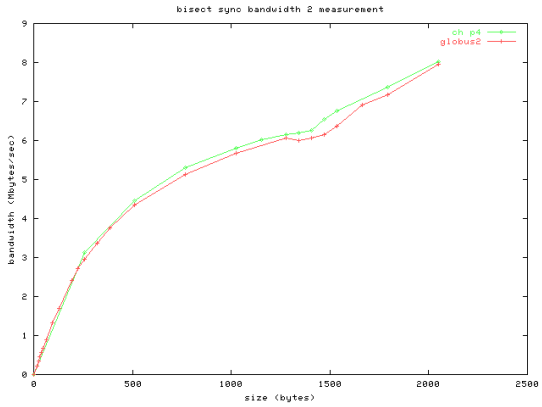


Figure 150: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 2 kbytes

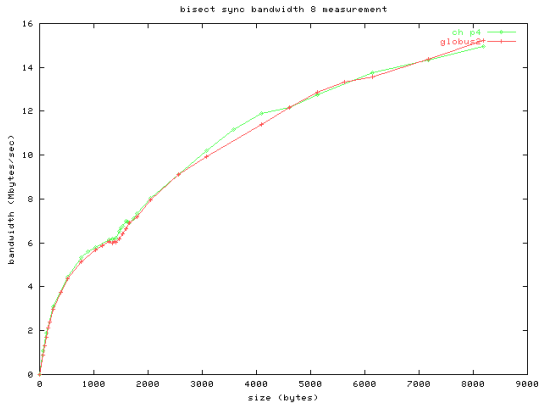


Figure 151: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 8 kbytes

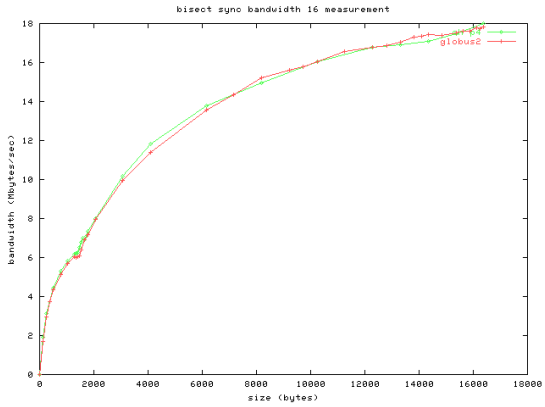


Figure 152: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 16 kbytes

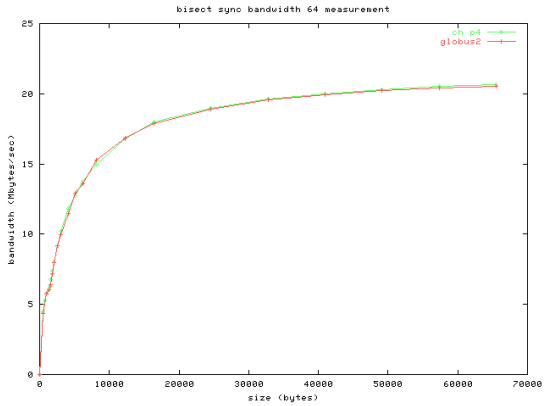


Figure 153: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 64 kbytes

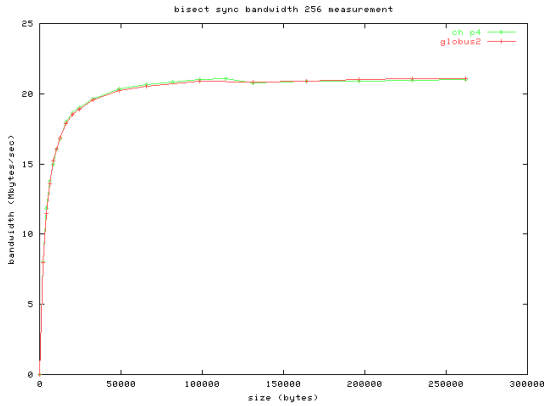


Figure 154: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 256 kbytes

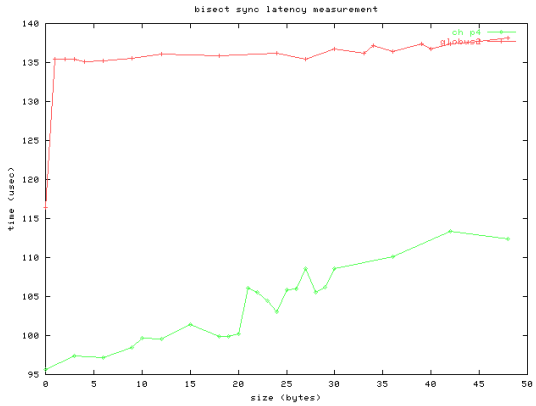


Figure 155: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 50 bytes

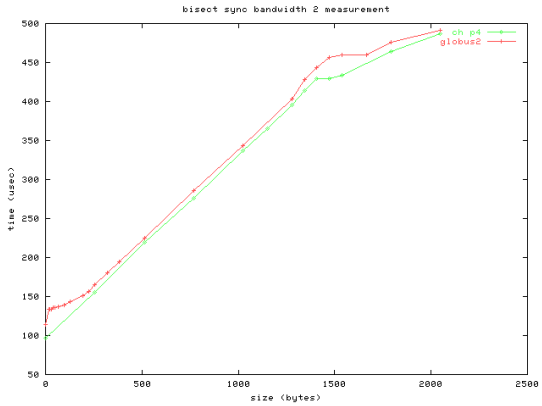


Figure 156: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 2 kbytes

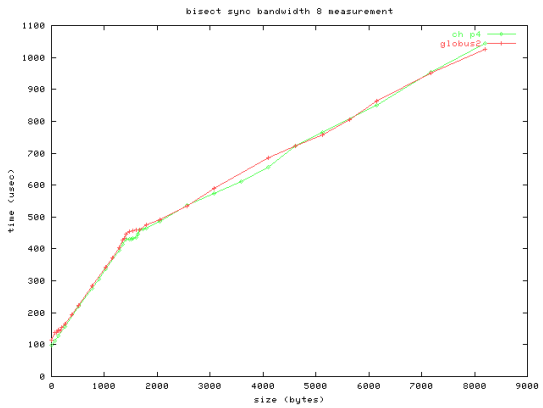


Figure 157: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 8 kbytes

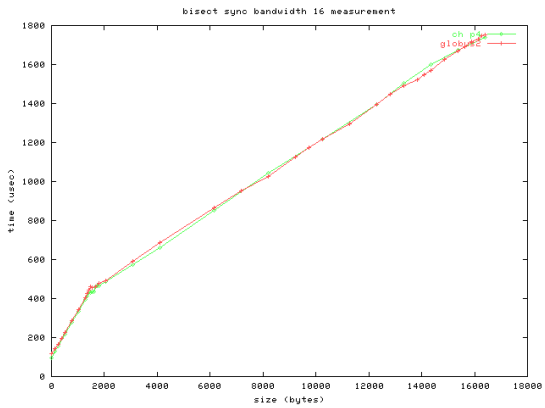


Figure 158: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 16 kbytes

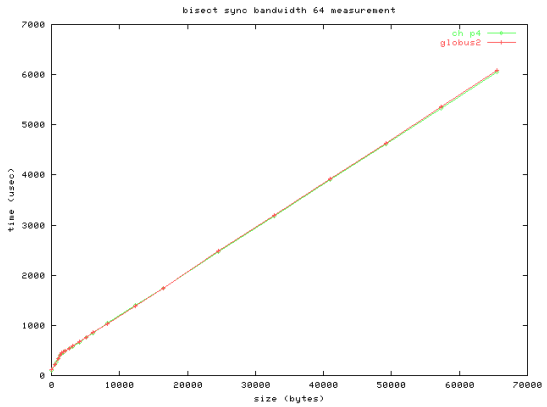


Figure 159: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 64 kbytes

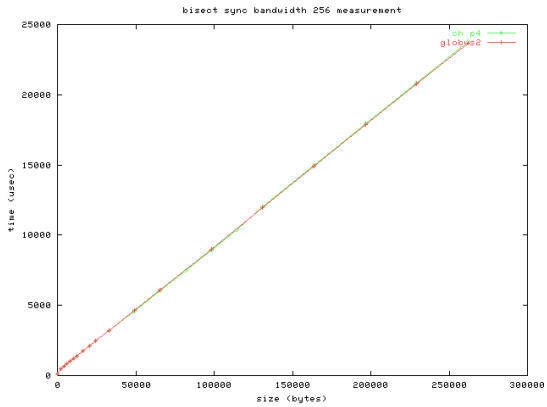


Figure 160: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 256 kbytes

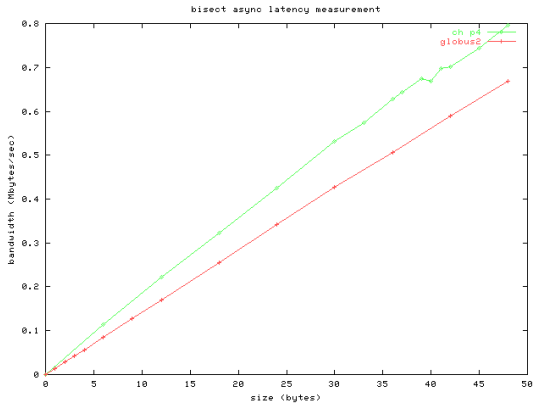


Figure 161: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 50 bytes

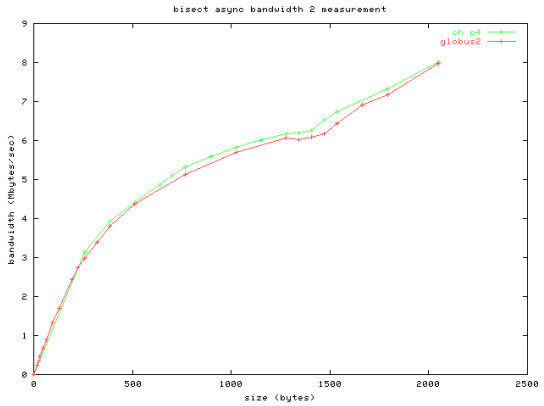


Figure 162: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 2 kbytes

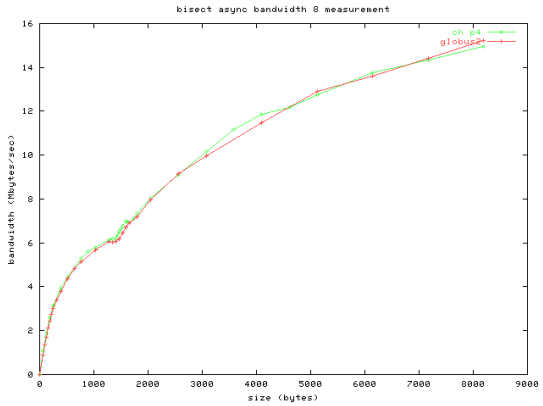


Figure 163: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 8 kbytes

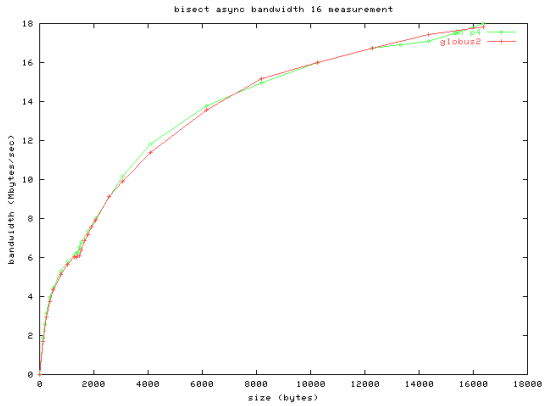


Figure 164: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 16 kbytes

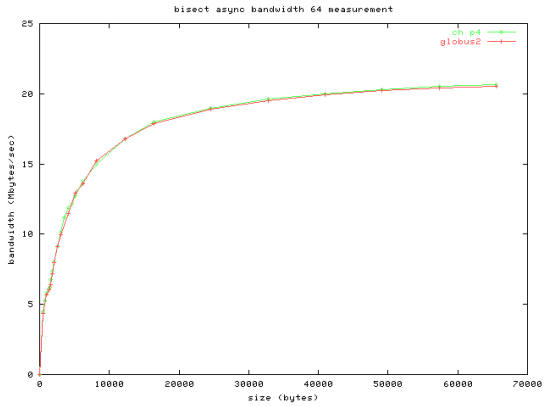


Figure 165: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 64 kbytes

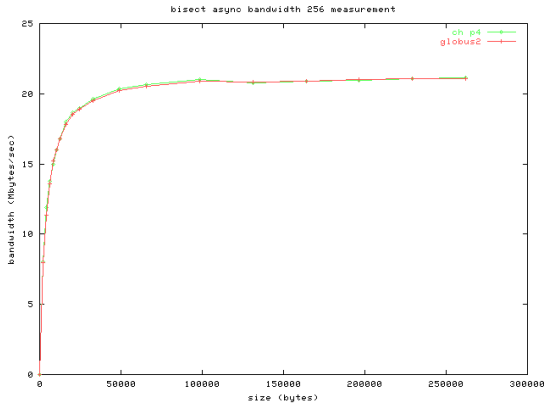


Figure 166: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 256 kbytes

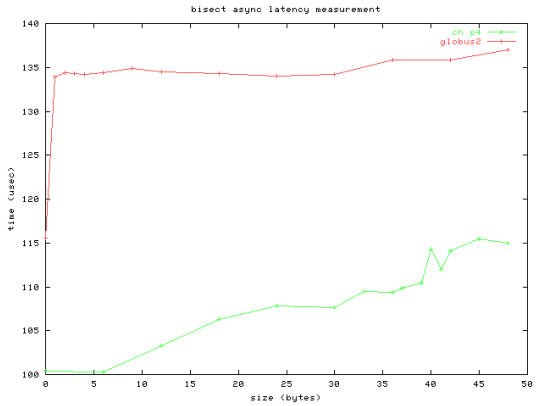


Figure 167: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 50 bytes

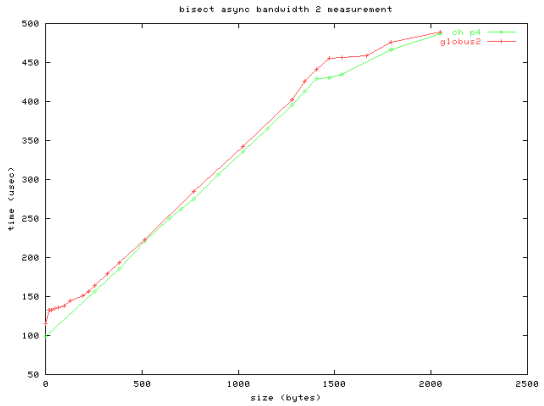


Figure 168: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 2 kbytes

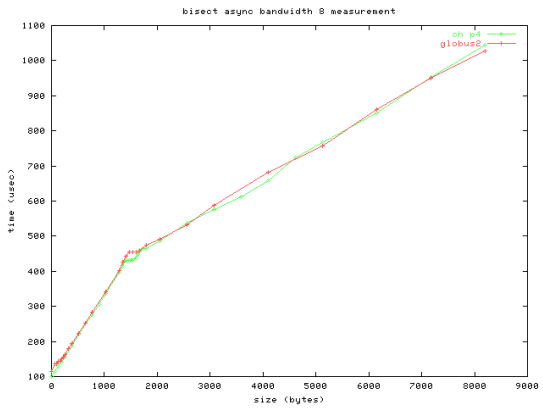


Figure 169: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 8 kbytes

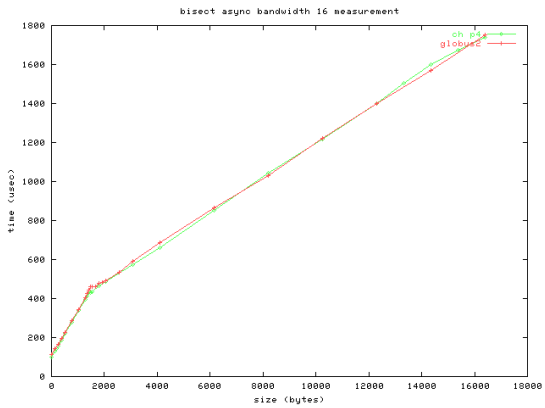


Figure 170: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 16 kbytes

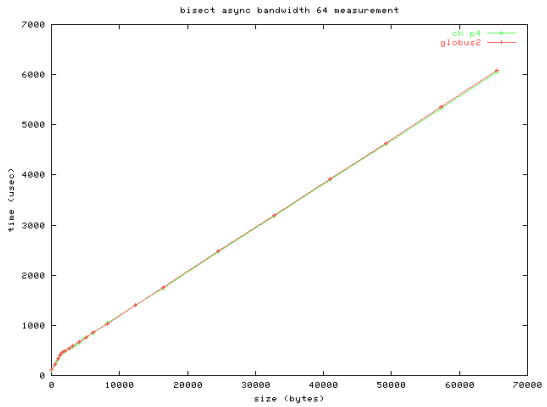


Figure 171: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 64 kbytes

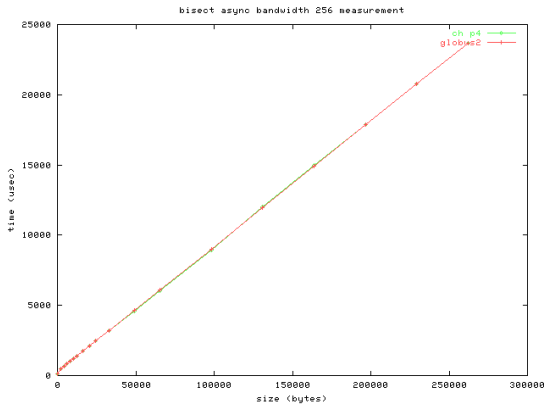


Figure 172: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 256 kbytes

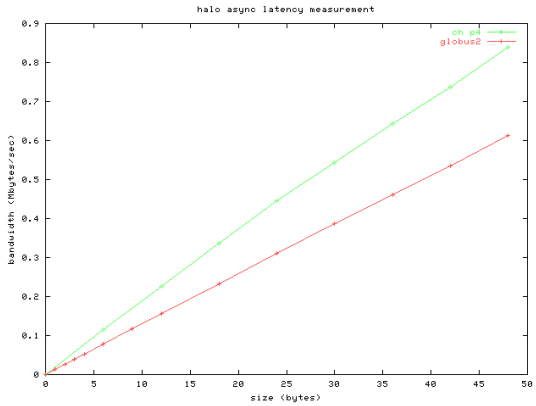


Figure 173: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 50 bytes

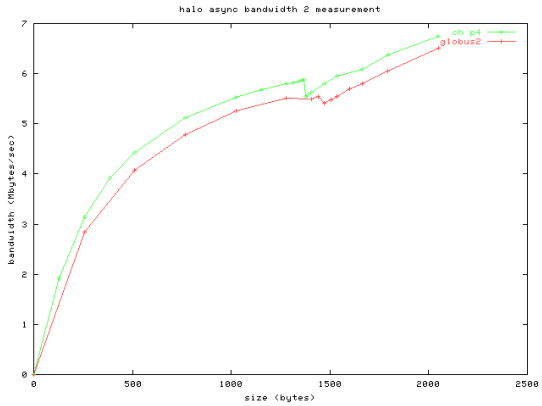


Figure 174: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 2 kbytes

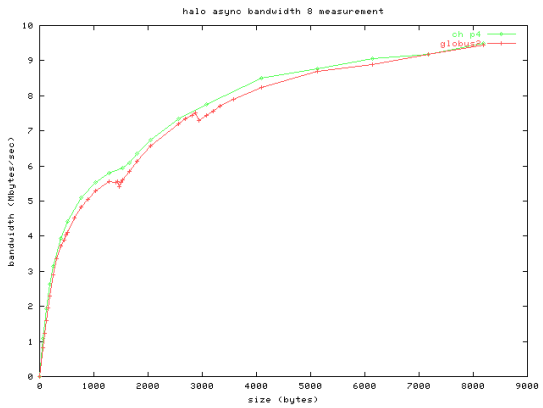


Figure 175: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 8 kbytes

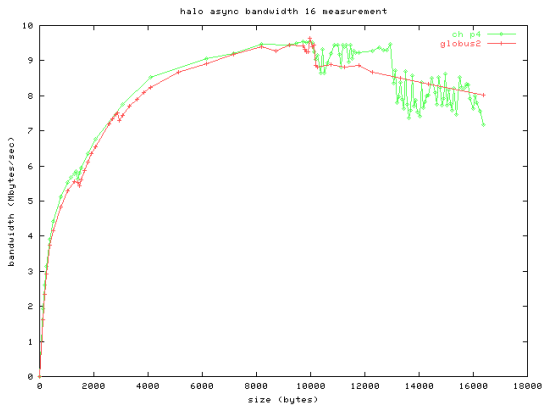


Figure 176: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 16 kbytes

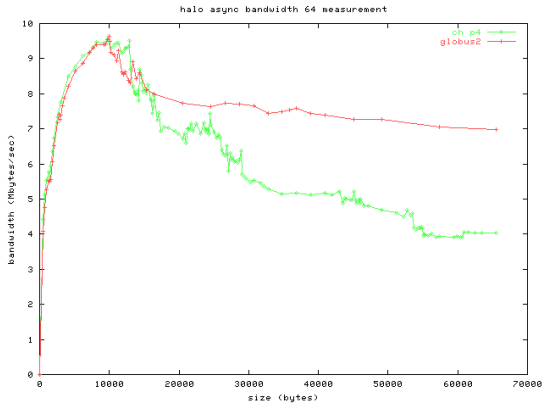


Figure 177: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 64 kbytes

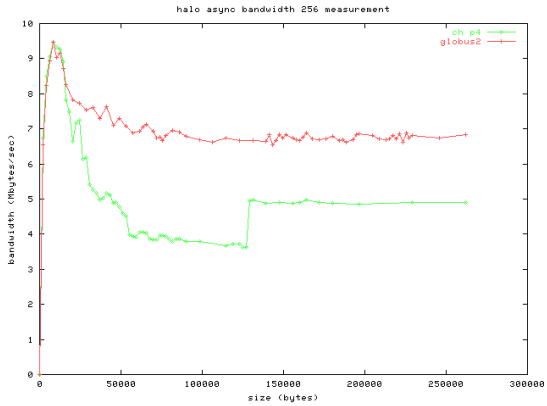


Figure 178: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 256 kbytes

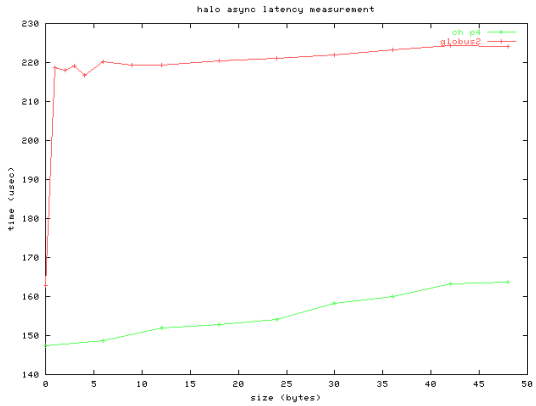


Figure 179: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 50 bytes

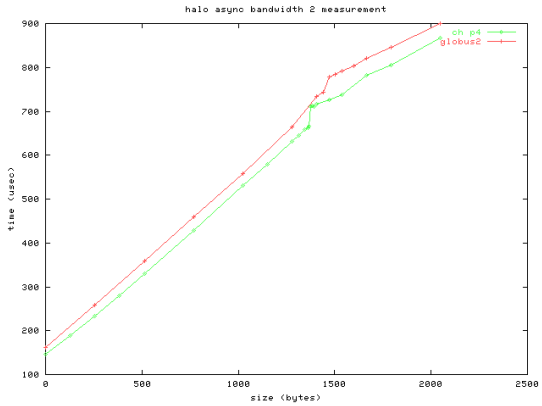


Figure 180: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 2 kbytes

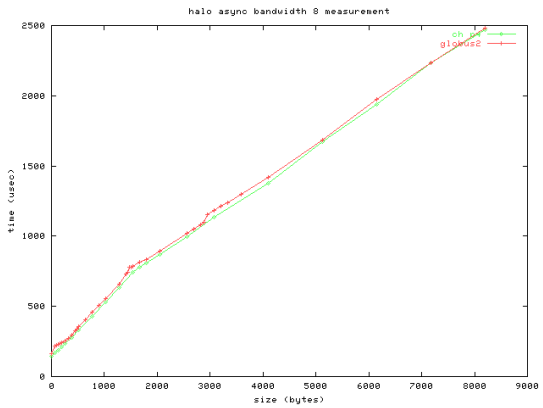


Figure 181: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 8 kbytes

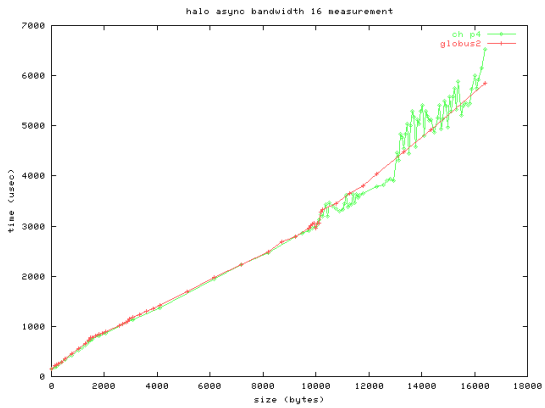


Figure 182: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 16 kbytes

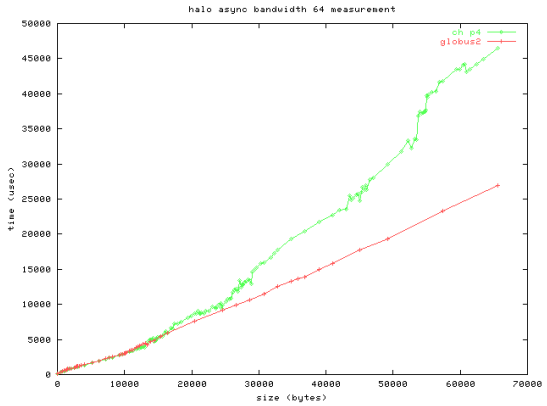


Figure 183: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 64 kbytes

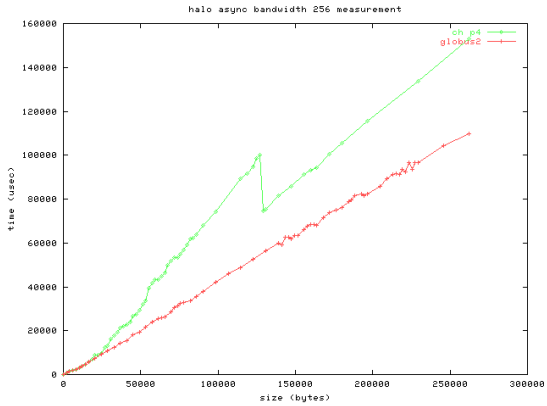


Figure 184: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 256 kbytes

B.4 Point-to-Point Tests on Cluster Level with eight Processes

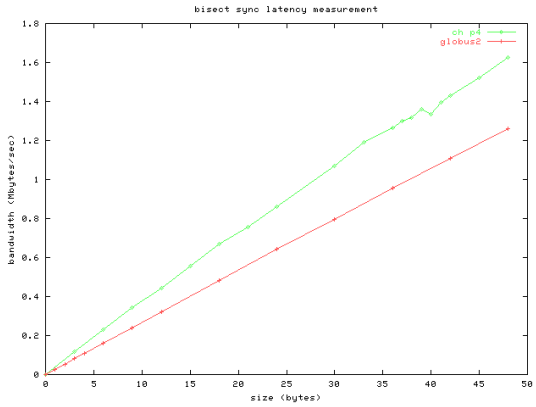


Figure 185: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 50 bytes

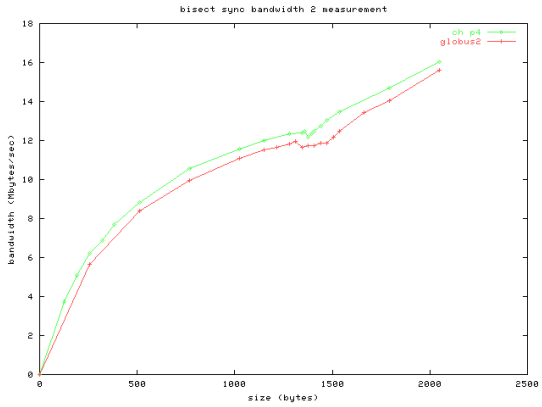


Figure 186: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 2 kbytes

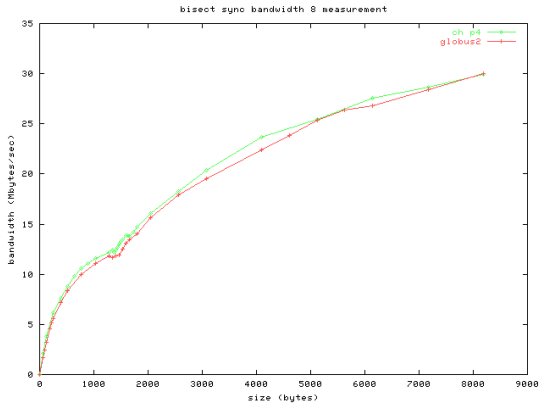


Figure 187: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 8 kbytes

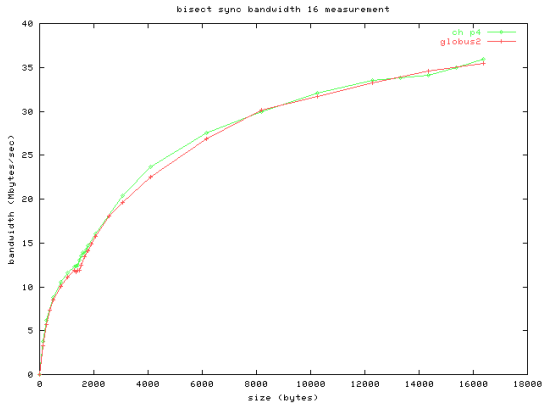


Figure 188: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 16 kbytes

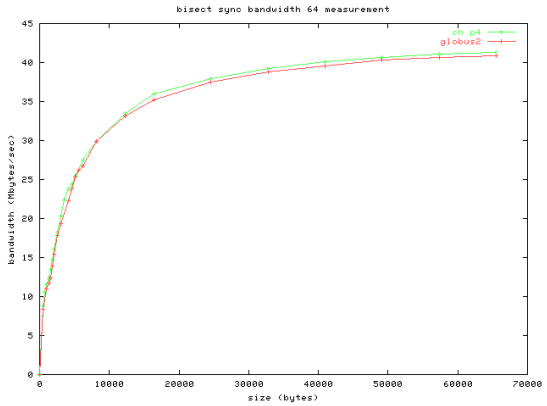


Figure 189: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 64 kbytes

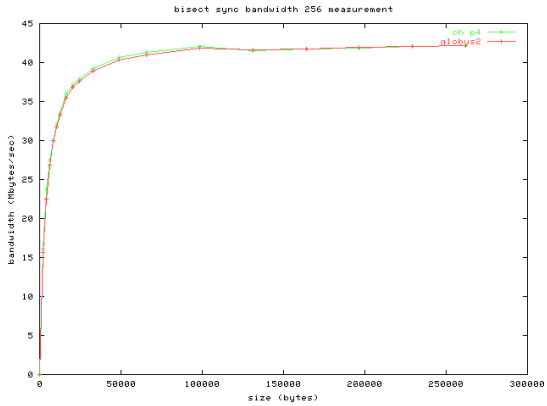


Figure 190: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 256 kbytes

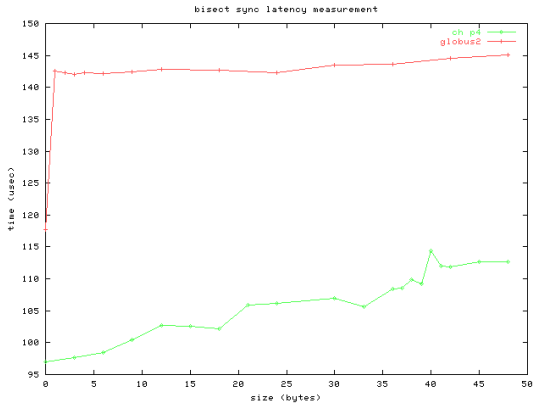


Figure 191: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 50 bytes

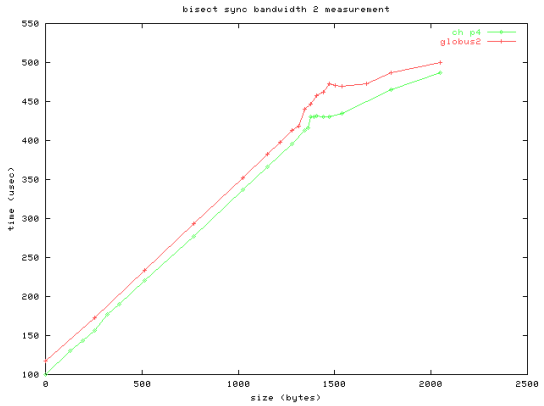


Figure 192: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 2 kbytes

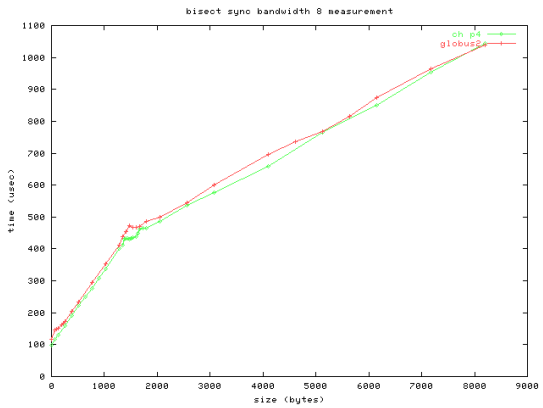


Figure 193: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 8 kbytes

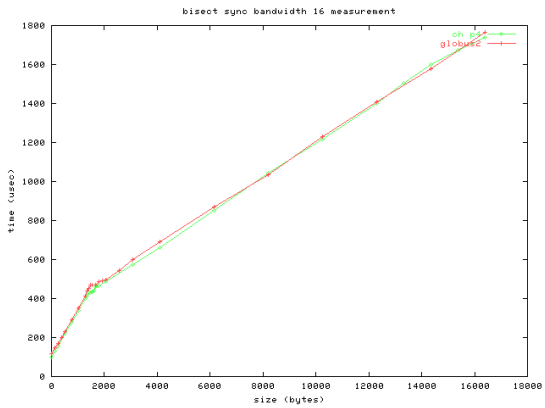


Figure 194: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 16 kbytes

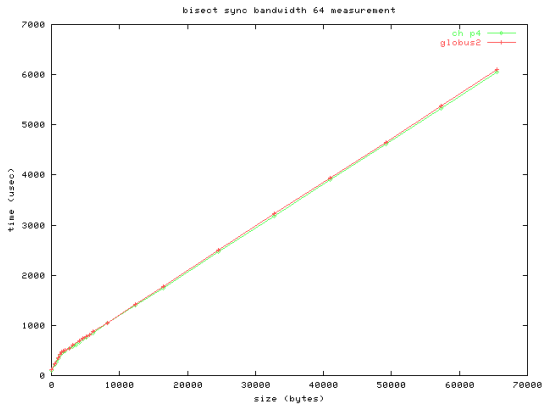


Figure 195: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 64 kbytes

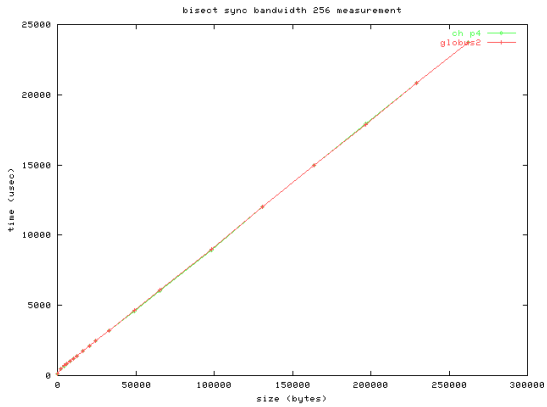


Figure 196: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 256 kbytes

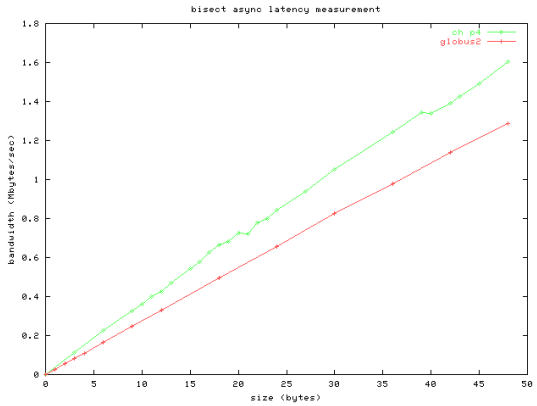


Figure 197: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 50 bytes

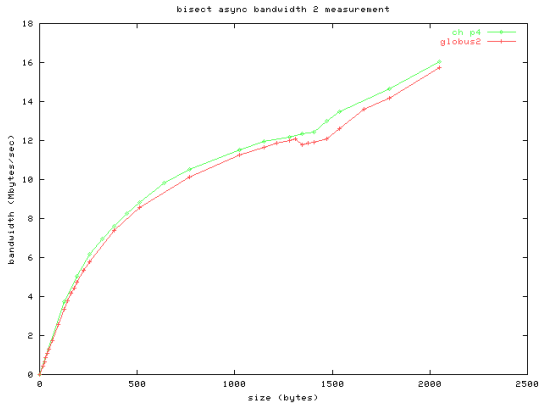


Figure 198: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 2 kbytes

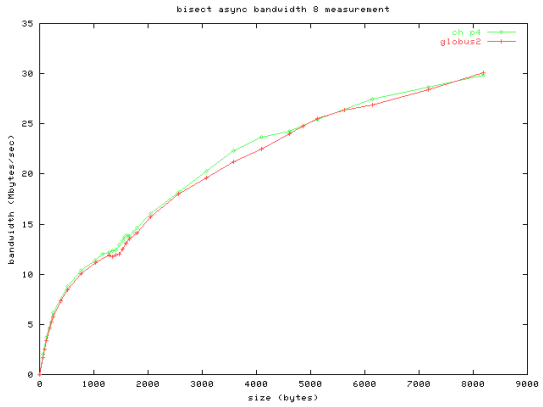


Figure 199: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 8 kbytes

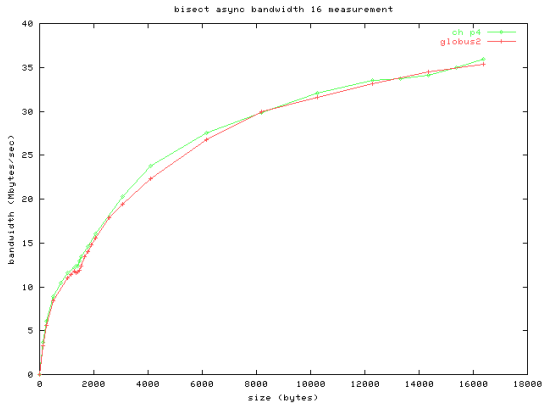


Figure 200: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 16 kbytes

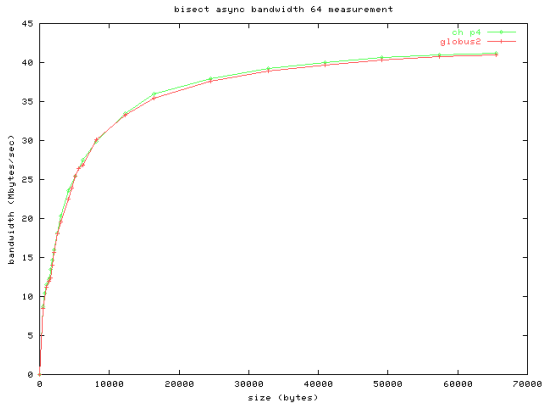


Figure 201: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 64 kbytes

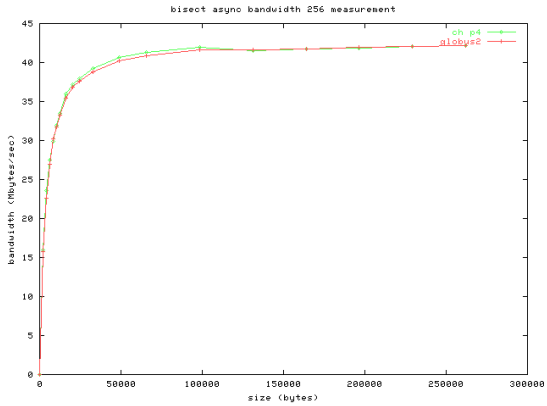


Figure 202: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 256 kbytes

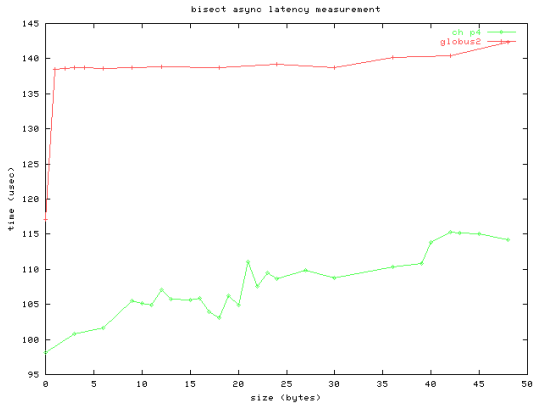


Figure 203: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 50 bytes

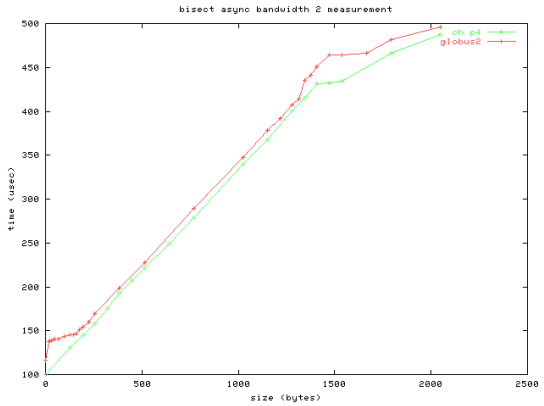


Figure 204: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 2 kbytes

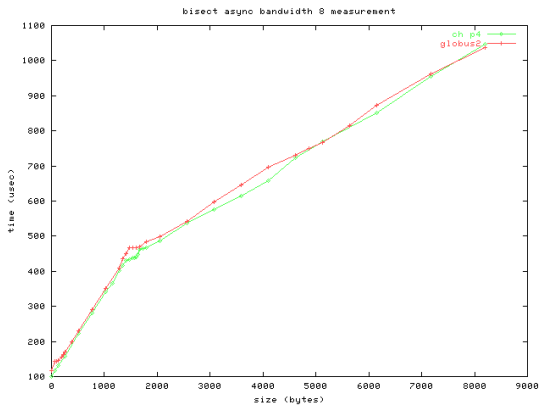


Figure 205: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 8 kbytes

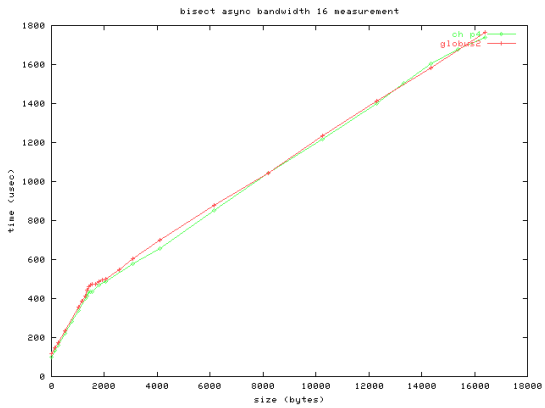


Figure 206: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 16 kbytes

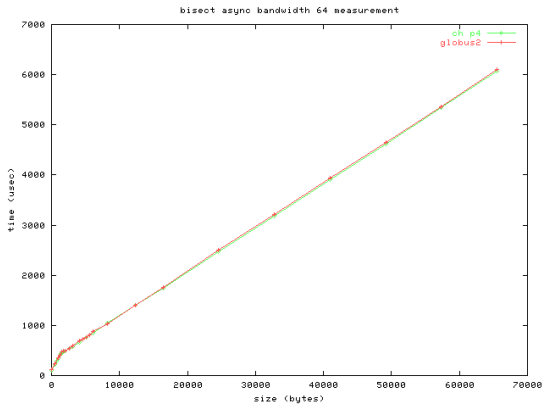


Figure 207: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 64 kbytes

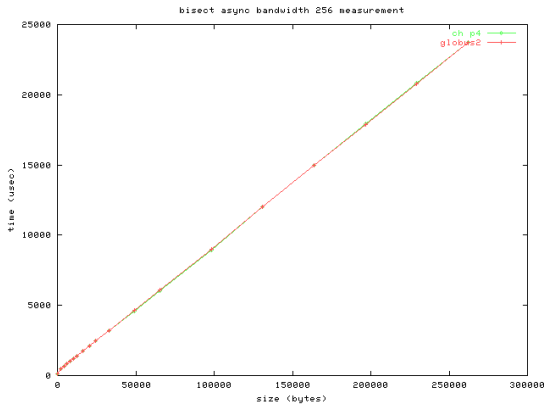


Figure 208: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 256 kbytes

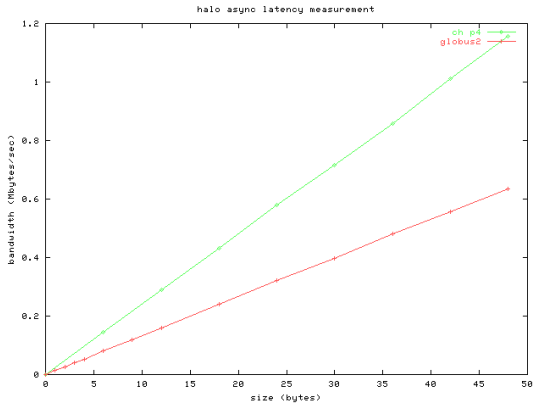


Figure 209: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 50 bytes

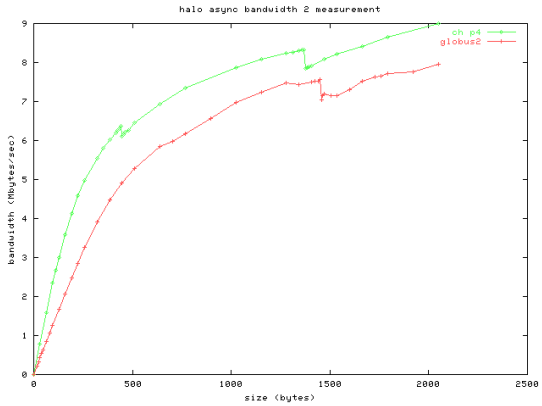


Figure 210: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 2 kbytes

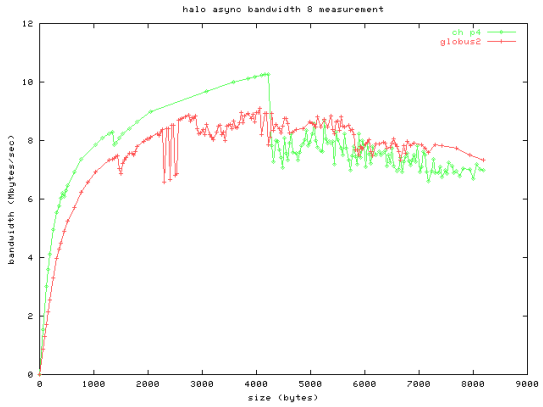


Figure 211: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 8 kbytes

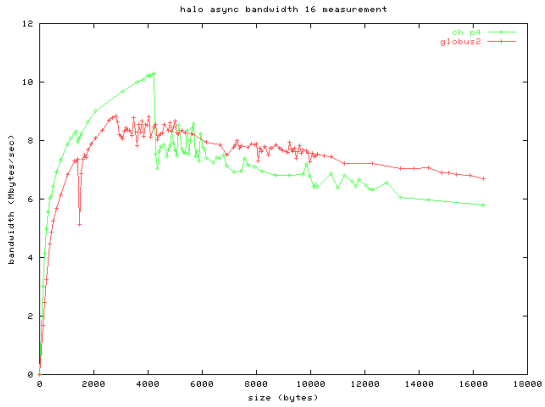


Figure 212: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 16 kbytes

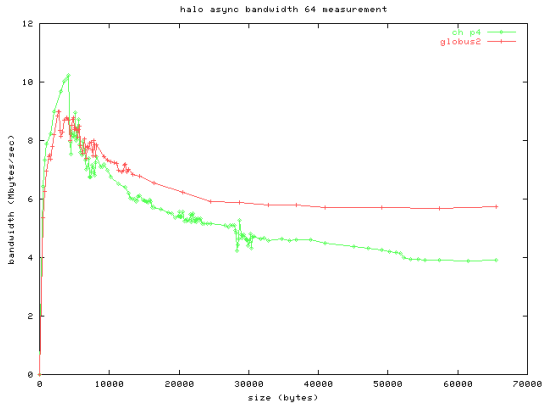


Figure 213: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 64 kbytes

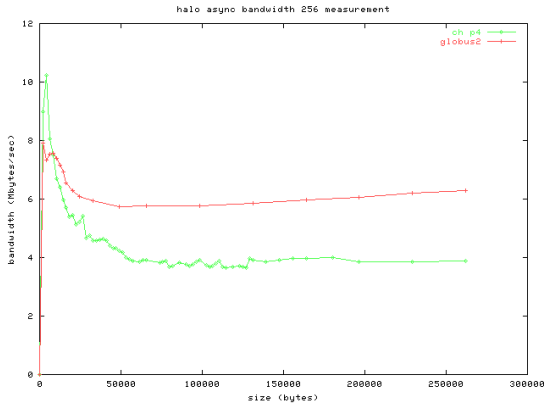


Figure 214: LAN level: Pt2pt (halo, non-blocking) bandwidth, max. msg size 256 kbytes

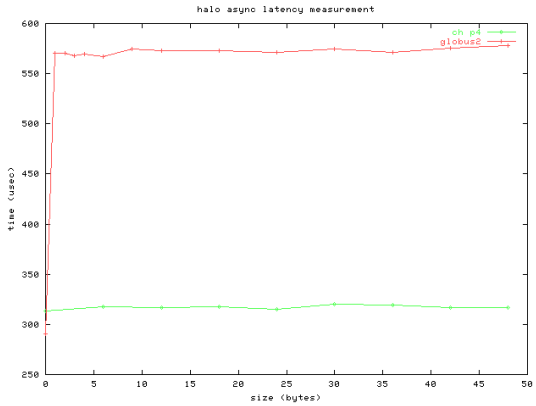


Figure 215: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 50 bytes

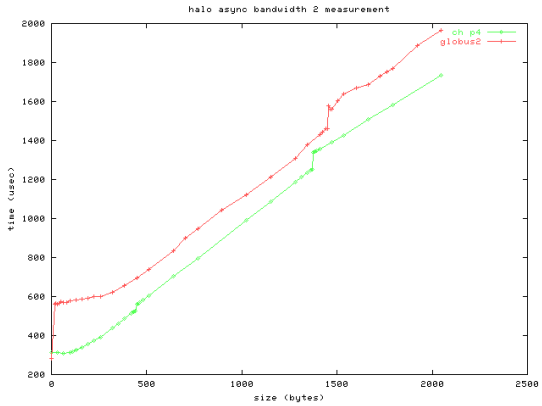


Figure 216: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 2 kbytes

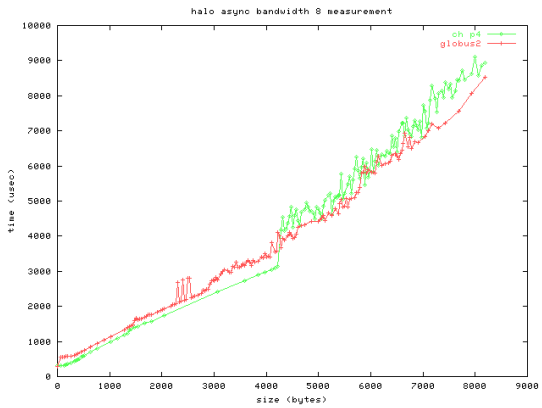


Figure 217: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 8 kbytes

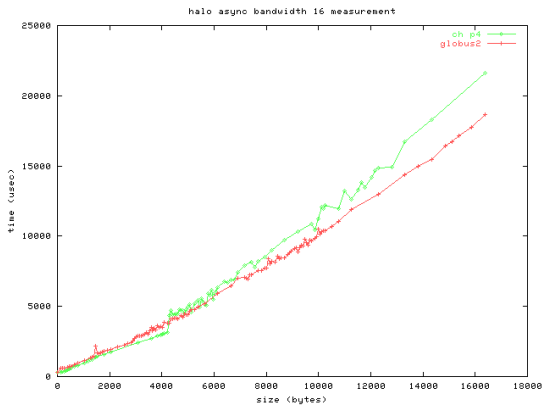


Figure 218: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 16 kbytes

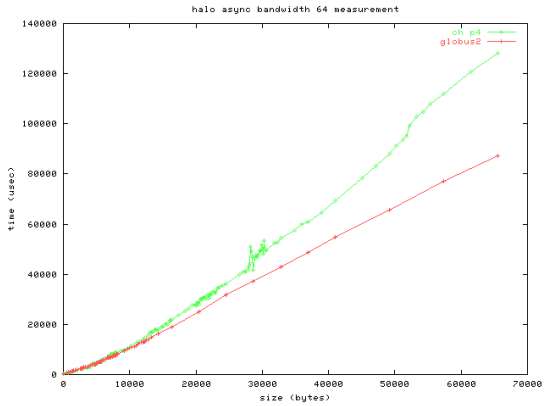


Figure 219: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 64 kbytes

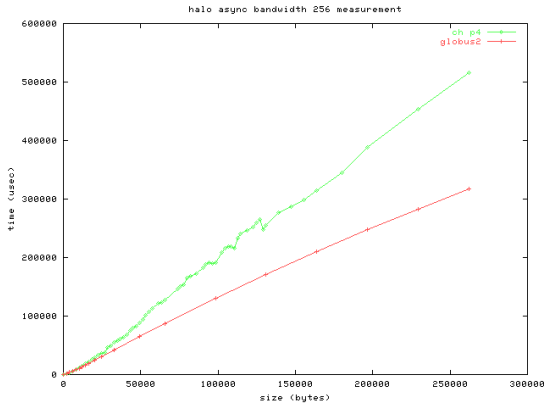


Figure 220: LAN level: Pt2pt (halo, non-blocking) timing, max. msg size 256 kbytes

B.5 Point-to-Point Tests on Cluster Level with 16 Processes

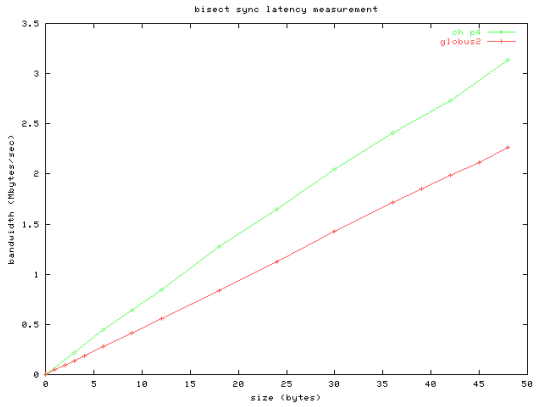


Figure 221: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 50 bytes

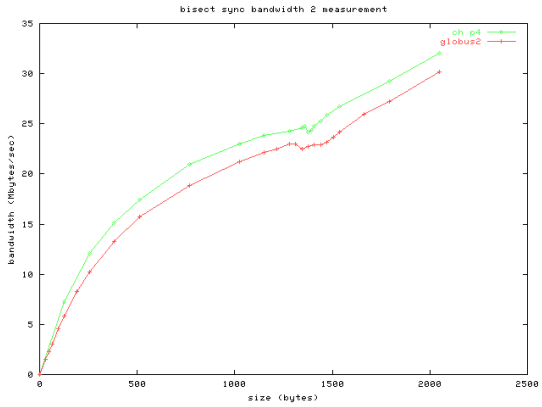


Figure 222: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 2 kbytes

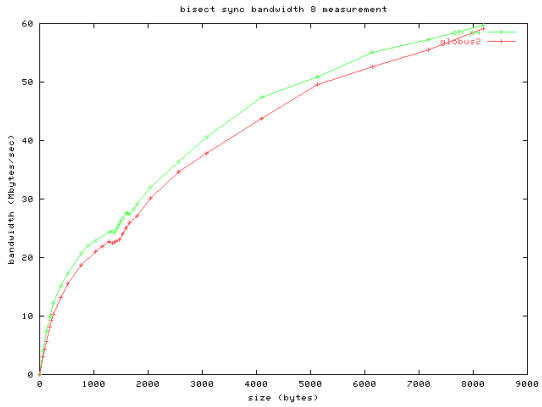


Figure 223: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 8 kbytes

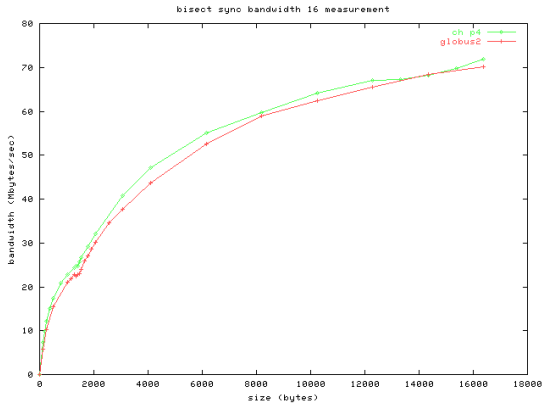


Figure 224: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 16 kbytes

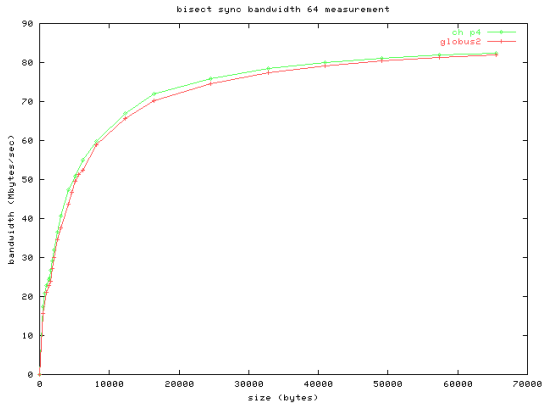


Figure 225: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 64 kbytes

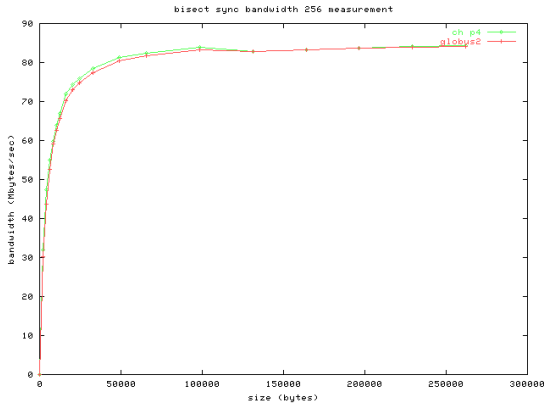


Figure 226: LAN level: Pt2pt (bisection, blocking) bandwidth, max. msg size 256 kbytes

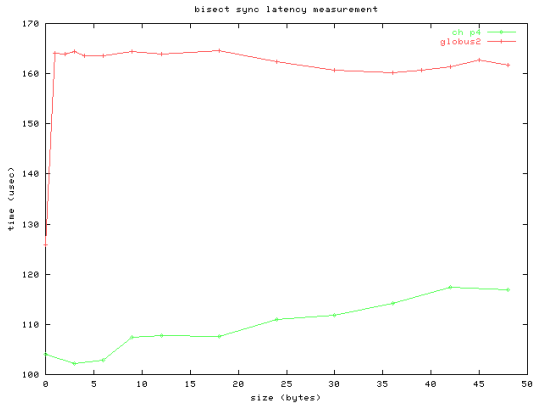


Figure 227: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 50 bytes

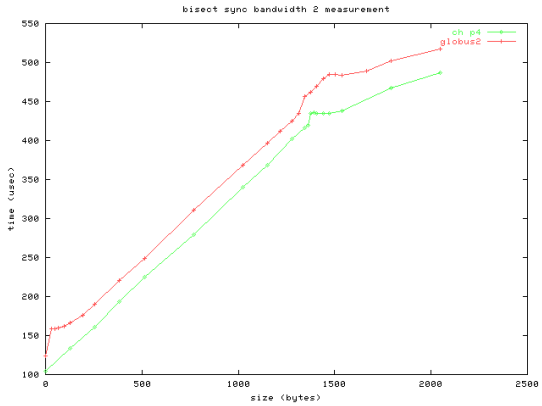


Figure 228: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 2 kbytes

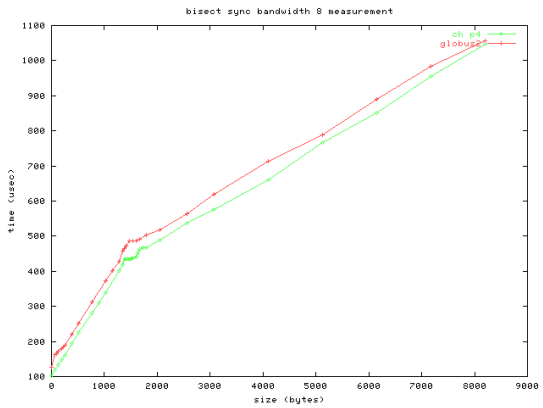


Figure 229: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 8 kbytes

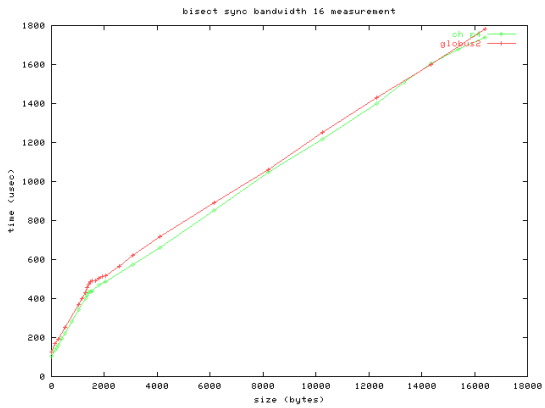


Figure 230: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 16 kbytes

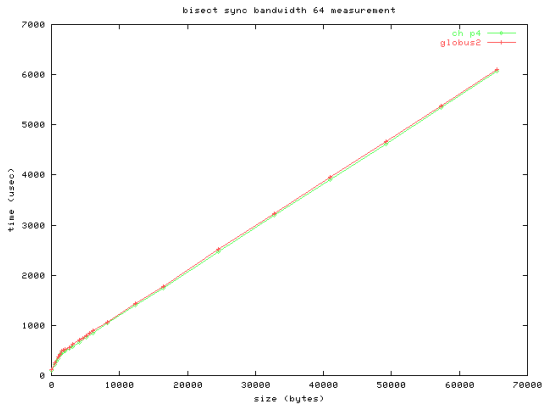


Figure 231: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 64 kbytes

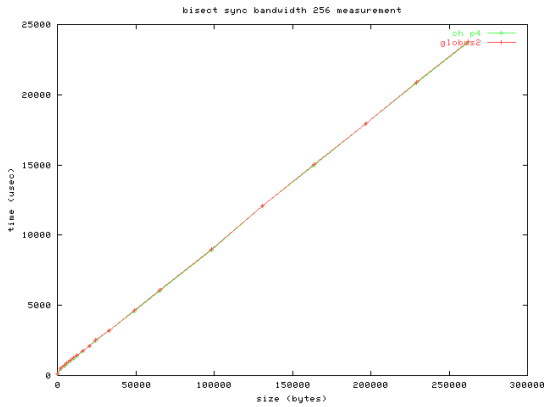


Figure 232: LAN level: Pt2pt (bisection, blocking) timing, max. msg size 256 kbytes

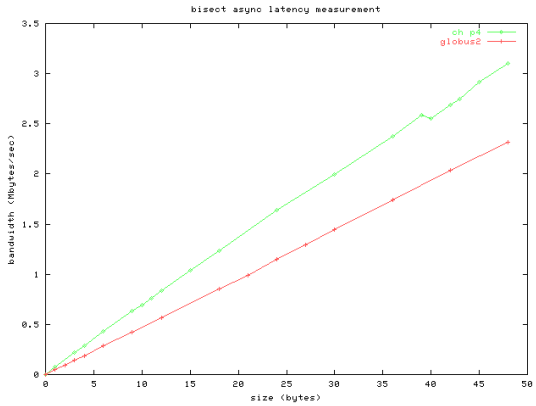


Figure 233: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 50 bytes

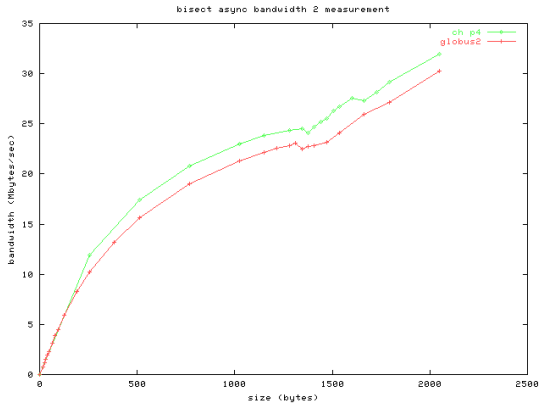


Figure 234: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 2 kbytes

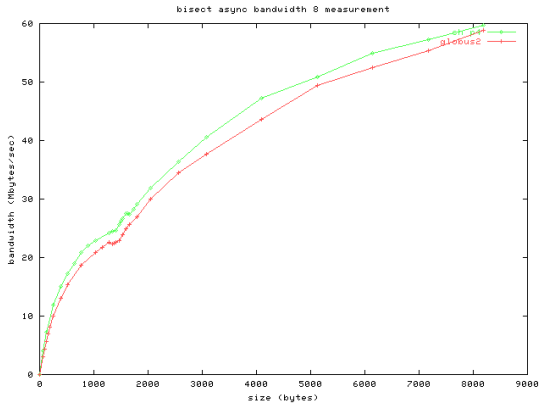


Figure 235: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 8 kbytes

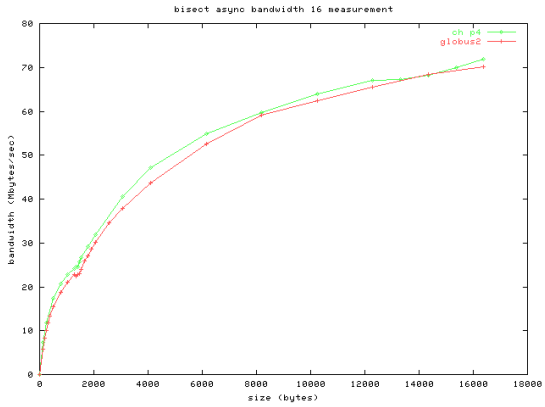


Figure 236: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 16 kbytes

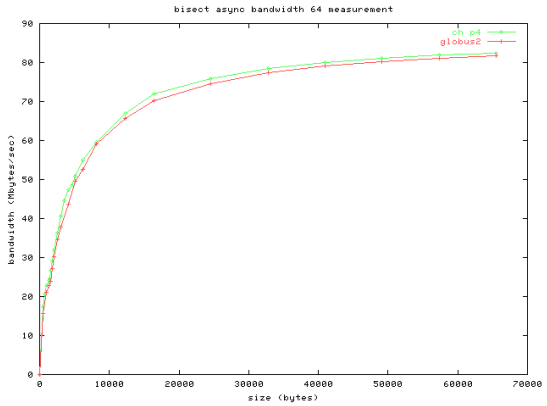


Figure 237: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 64 kbytes

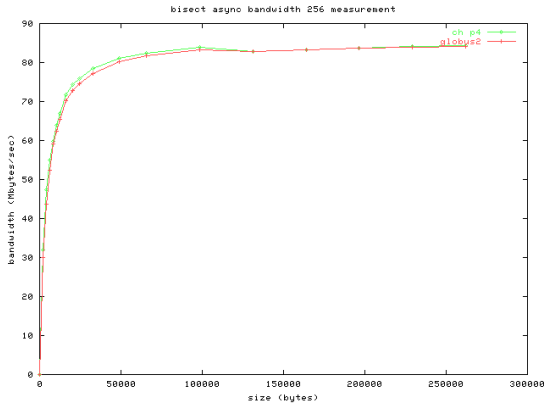


Figure 238: LAN level: Pt2pt (bisection, non-blocking) bandwidth, max. msg size 256 kbytes

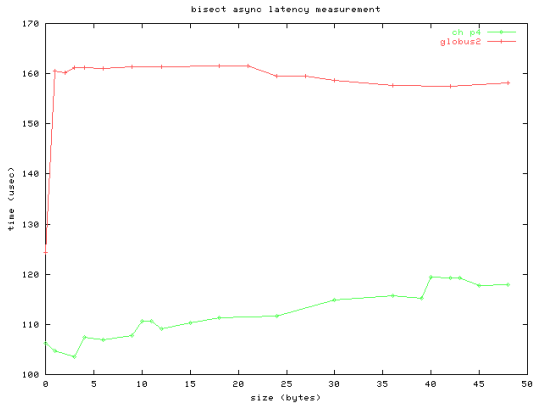


Figure 239: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 50 bytes

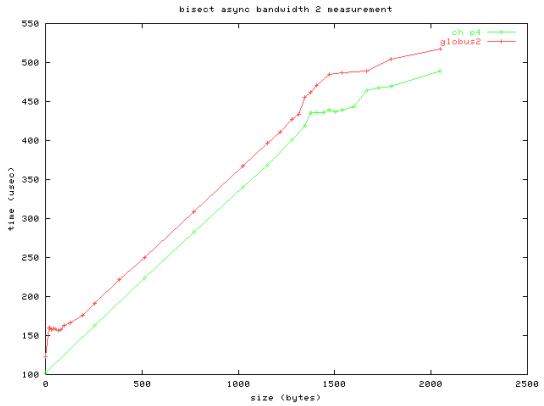


Figure 240: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 2 kbytes

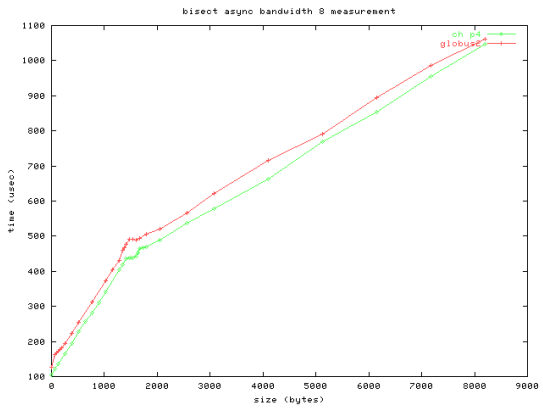


Figure 241: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 8 kbytes

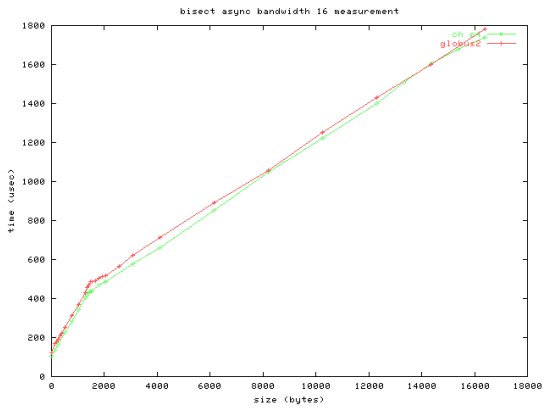


Figure 242: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 16 kbytes

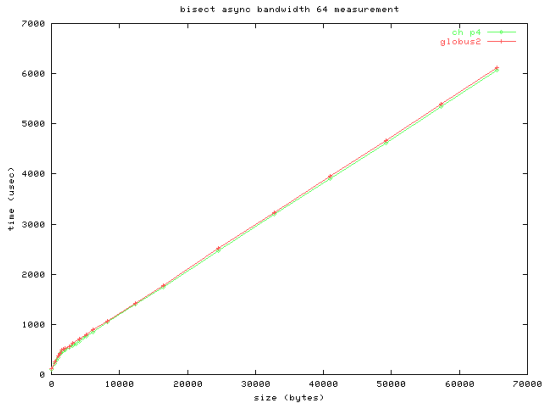


Figure 243: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 64 kbytes

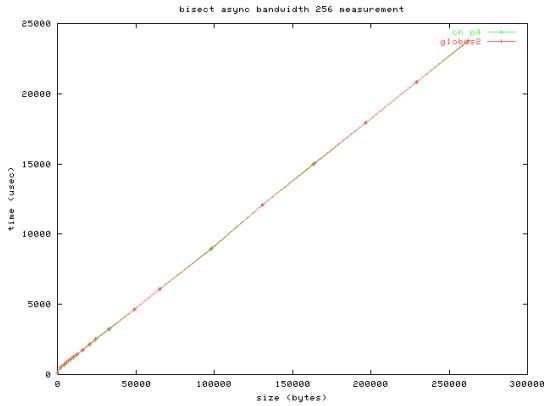


Figure 244: LAN level: Pt2pt (bisection, non-blocking) timing, max. msg size 256 kbytes

B.6 Collective Tests on Cluster-of-Clusters Level with 16 processes

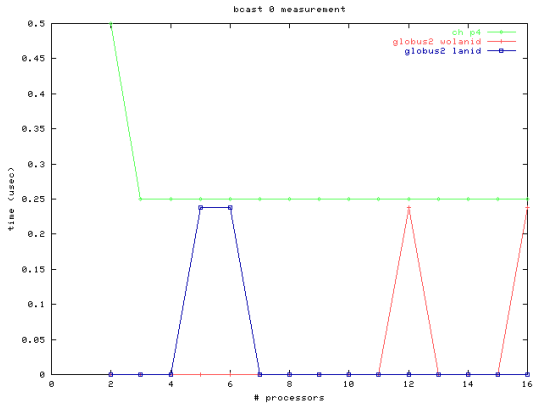


Figure 245: CoC level: Coll. oparation: broadcast, msg size 0 byte

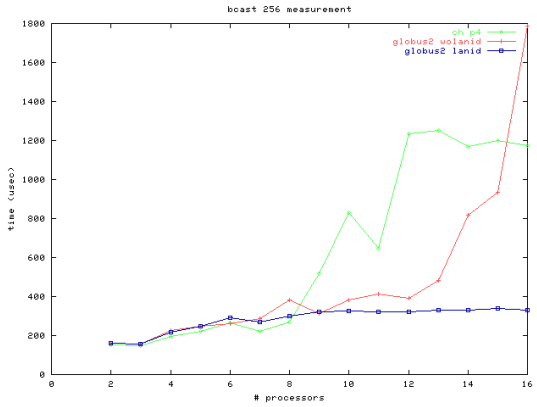


Figure 246: CoC level: Coll. oparation: broadcast, msg size 256 bytes

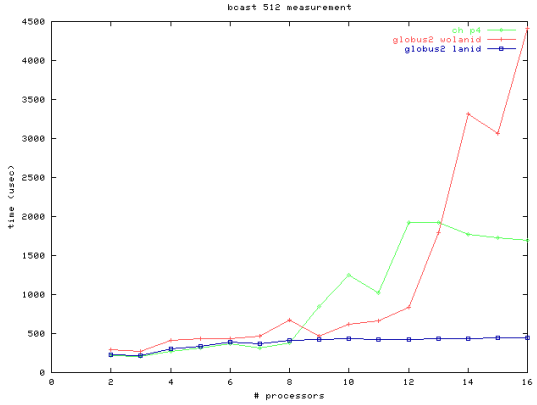


Figure 247: CoC level: Coll. oparation: broadcast, msg size 512 bytes

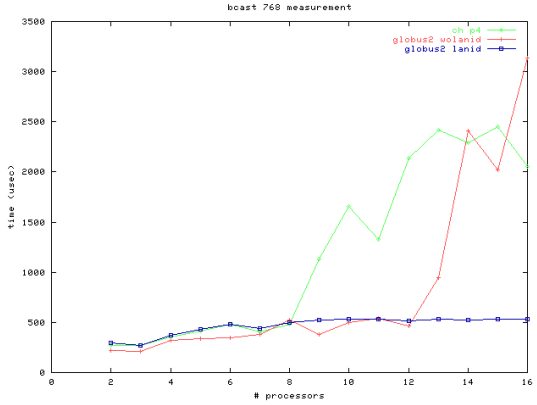


Figure 248: CoC level: Coll. oparation: broadcast, msg size 768 bytes

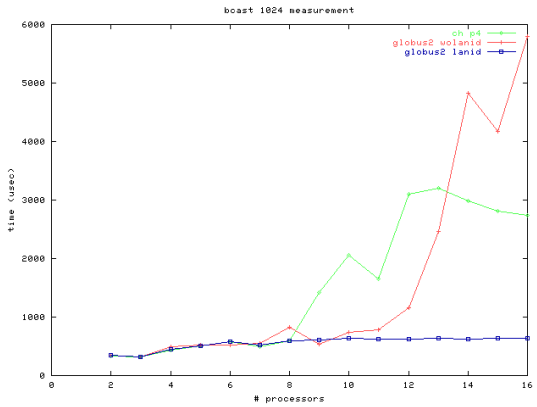


Figure 249: CoC level: Coll. oparation: broadcast, msg size 1024 bytes

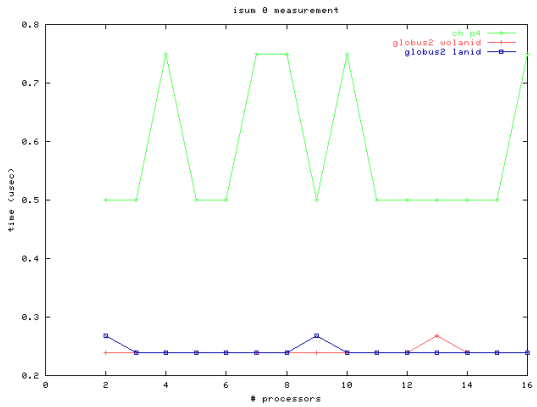


Figure 250: CoC level: Coll. operation: integer reduction, msg size 0 byte

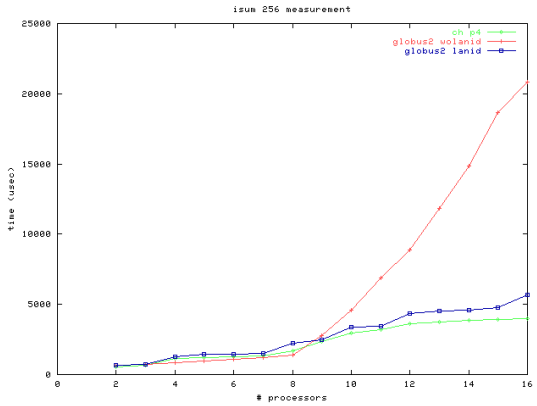


Figure 251: CoC level: Coll. operation: integer reduction, msg size 256 bytes

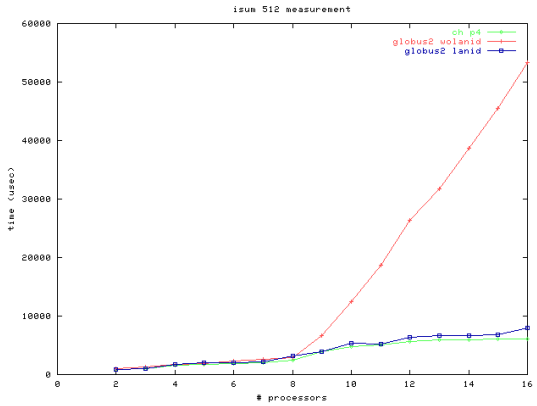


Figure 252: CoC level: Coll. operation: integer reduction, msg size 512 bytes

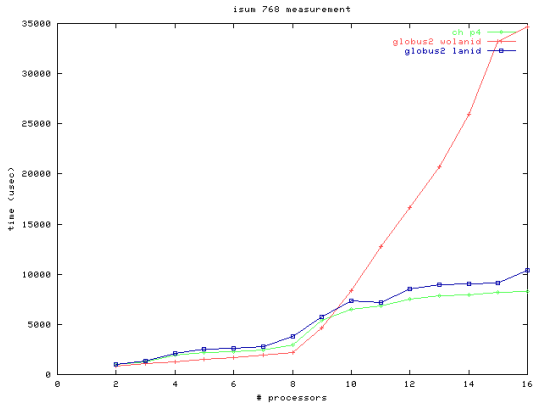


Figure 253: CoC level: Coll. operation: integer reduction, msg size 768 bytes

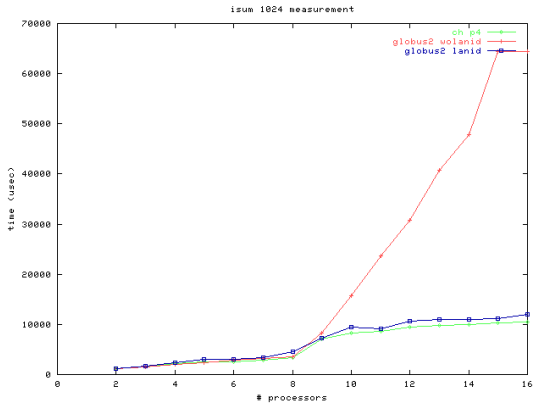


Figure 254: CoC level: Coll. operation: integer reduction, msg size 1024 bytes

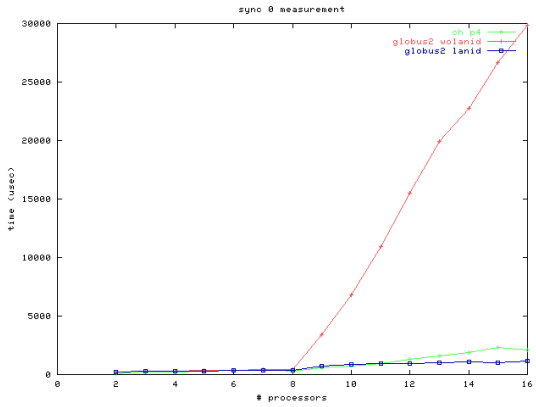


Figure 255: CoC level: Coll. operation: synchronization

B.7 Collective Tests on Cluster-of-Clusters Level with 32 processes

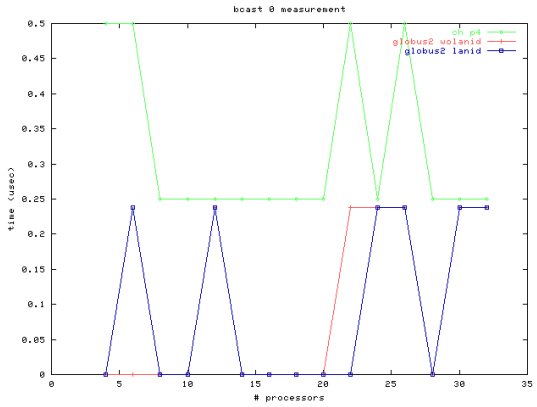


Figure 256: CoC level: Coll. oparation: broadcast, msg size 0 byte

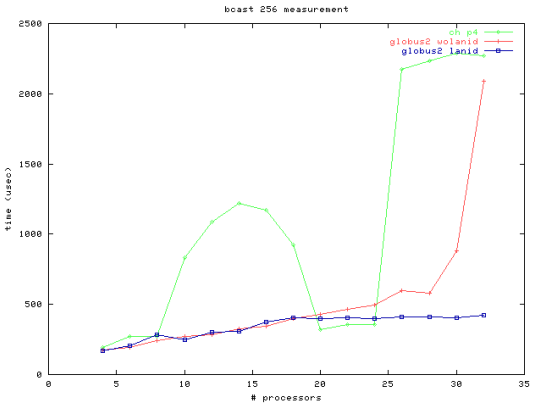


Figure 257: CoC level: Coll. oparation: broadcast, msg size 256 bytes

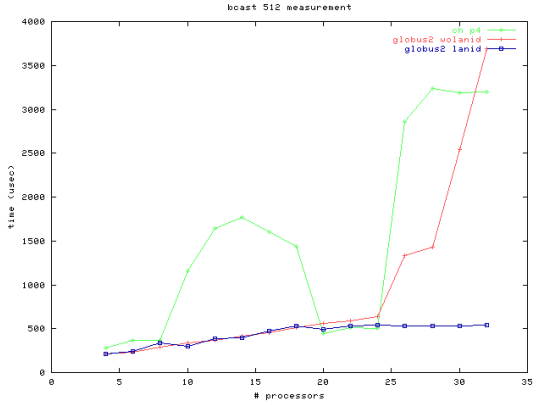


Figure 258: CoC level: Coll. oparation: broadcast, msg size 512 bytes

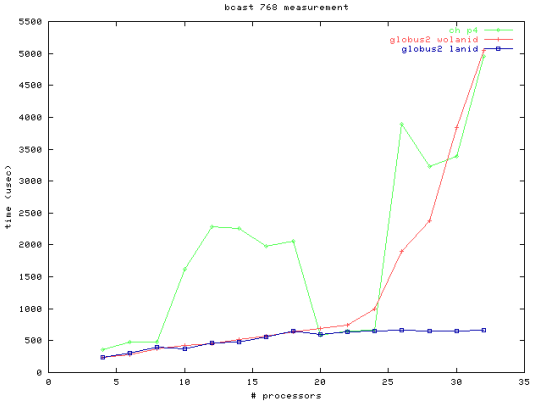


Figure 259: CoC level: Coll. oparation: broadcast, msg size 768 bytes

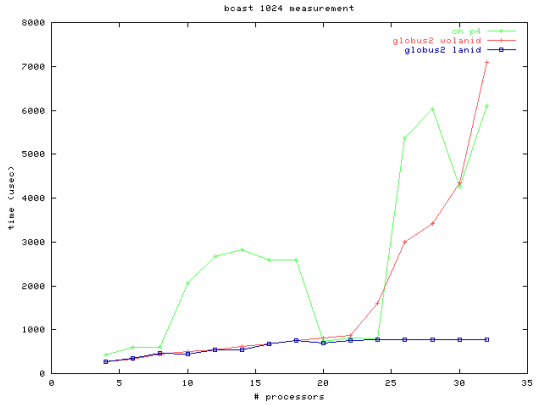


Figure 260: CoC level: Coll. oparation: broadcast, msg size 1024 bytes

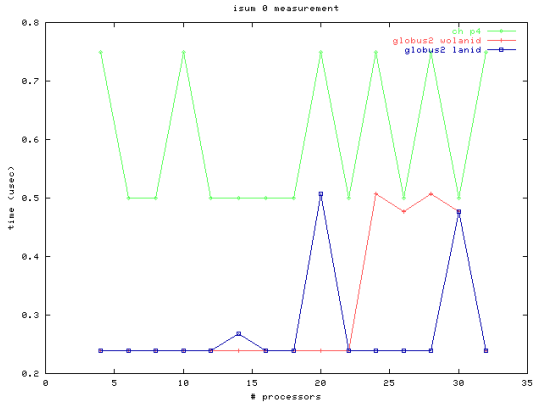


Figure 261: CoC level: Coll. operation: integer reduction, msg size 0 byte

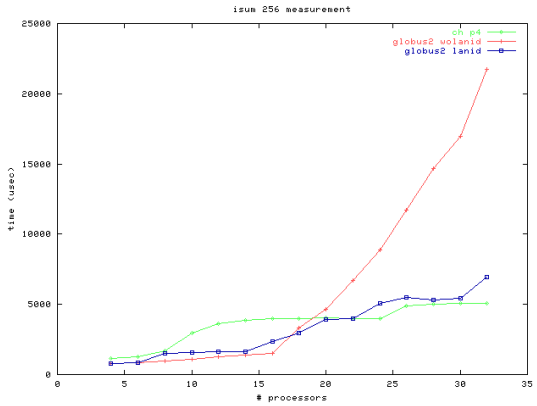


Figure 262: CoC level: Coll. operation: integer reduction, msg size 256 bytes

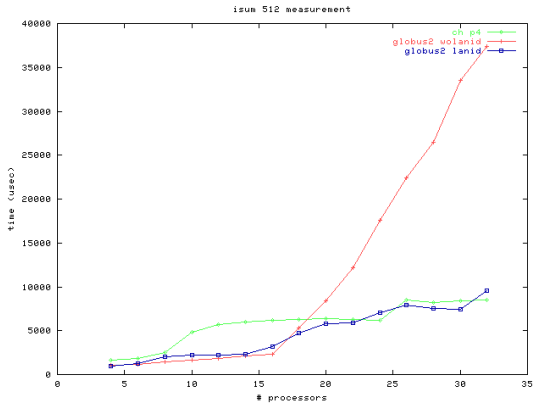


Figure 263: CoC level: Coll. operation: integer reduction, msg size 512 bytes

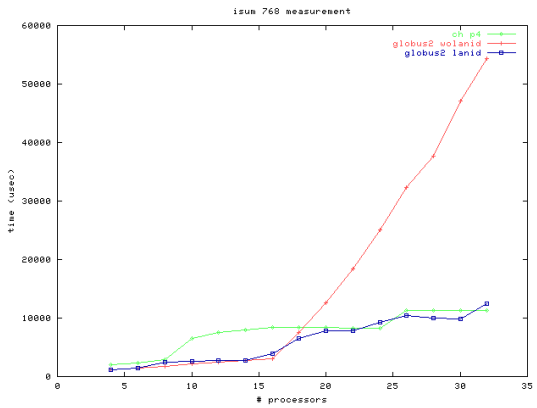


Figure 264: CoC level: Coll. operation: integer reduction, msg size 768 bytes

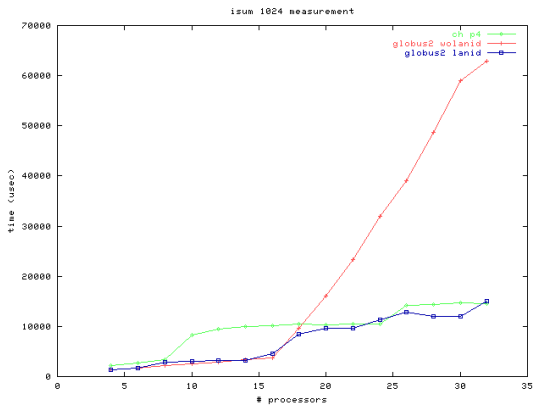


Figure 265: CoC level: Coll. operation: integer reduction, msg size 1024 bytes

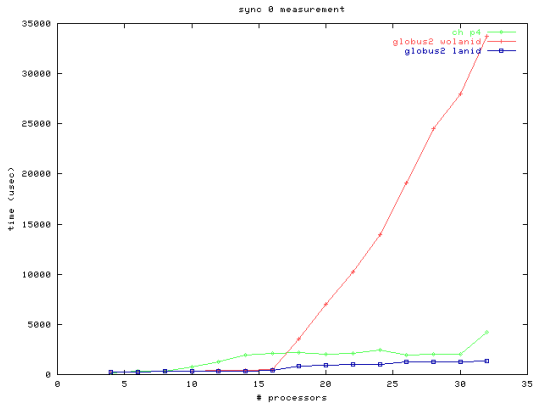


Figure 266: CoC level: Coll. operation: synchronization

List of Figures

List of Tables

References

[Allan01] *Allan, R.J., Hanlon, D., Smith, G., Fowler, R.F., Greenough, C.:* A Globus Developers' Guide with Installation and Maintenance Hints. Draft from 14 September 2001, http://esc.dl.ac.uk/StarterKit/PS/globus_guide.ps, fetch on 2002/07/12

[Beowu02] *Scyld Computing Corporation*, originally *CESDIS, NASA's Goddard Space Flight Center:* The Original Beowulf. http://beowulf.es.embnet.org/gsfc-hw.html, fetch on 2002/07/03

[Burge02] *Burgess, M.:* Configuration Engine. http://www.cfengine.org/, fetch on 2002/06/26

[Chan02] *Chan, A., Gropp, B., Lusk, R.:* Performance Visualization for Parallel Programs. LANS, labratory for advanced numerical software. http://www.mcs.anl.gov/perfvis/, fetch on 2002/09/23

[CTeam02] *Condor Development Team:* The Condor Project Homepage. http://www.cs.wisc.edu/condor/, fetch on 2002/06/26

[Foste02a] *Foster, I., Kesselman, C., Tuecke, S.:* The Anatomy of the Grid. Enabling Scalable Virtual Organizations. ,2002, http://www.globus.org/research/papers/anatomy.pdf, fetch on 2002/07/03

[Foste02b] *Foster, I.:* What is the Grid? A three point checklist. Daily news and informations for the global grid community, July 22, 2002: VOL. 1 NO. 6, http://www.gridtoday.com/02/0722/100136.html, fetch on 2002/09/10

[Foste99] *Foster, I., Kesselman, C.:* The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1998

[Frey01] *Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.:* Condor-G: A Computation Management Agent for Multi-Institutional Grids. http://www.cs.wisc.edu/condor/doc/condorg-hpdc10.pdf, fetch on 2002/06/26

[Grabn02] *Grabner, R.:* Condor. Hauptseminar 'Cluster- und Gridcomputing', Chemnitz University of Technology, 2002/02/05, http://archiv.tu-chemnitz.de/pub/2002/0076, fetch on 2002/06/26. *Documents available in German only.*

[Gropp02] *Gropp, W.:* How not to measure communications performance. http://www-unix.mcs.anl.gov/mpi/mpptest/hownot.html, fetch on 2002/07/24

[Gropp99] *Gropp, W., Lusk, E.:* Reproducible Measurements of MPI Performance Characteristics. http://www.mcs.anl.gov/ gropp/bib/papers/1999/pvmmpi99/mpptest.ps, fetch on 2002/07/24

[GTeam02a] *Globus Development Team:* Globus Toolkit 2.0. http://www.globus.org/toolkit, fetch on 2002/07/05

[GTeam02b] *Globus Development Team:* Globus Toolkit 2.0 Admin Guide. http://www.globus.org/gt2/admin/guide.html, fetch on 2002/06/20

[GTeam02c] *Globus Development Team:* Globus Toolkit 2.0 Installation Notes. http://www.globus.org/gt2/install, fetch on 2002/06/20

[GTeam02d] *Globus Development Team:* The Globus Resource Specification Language RSL v1.0. http://www.globus.org/gram/rsl_spec1.html, fetch on 2002/06/20

[GTeam02e] *Globus Development Team:* Globus Toolkit API Reference. http://www.globus.org/developer/api-reference.html, fetch on 2002/06/30

[IBMan02] *IBM announcement:* IBM Offers Commercial Support Of Revolutionary Grid Software. Daily news and informations for the global grid community, August 2, 2002, http://www.gridtoday.com/breaking/047.html, fetch on 2002/09/24

[MG2Te02] *MPICH-G2 Development Group:* MPICH-G2 Homepage. http://www.hpclab.niu.edu/mpi/, fetch on 2002/06/26

[MPI02] *MCS Division at Argonne National Laboratory:* Message Passing Interface. http://www-unix.mcs.anl.gov/mpi/, fetch on 2002/07/11

[Mpich02] *MPICH Development Group:* MPICH Homepage. http://www-unix.mcs.anl.gov/mpi/mpich/, fetch on 2002/06/26

[Palla00] *Pallas GmbH:* Pallas MPI Benchmarks - PMB, Part MPI-1, 2000/03/09, ftp://ftp.pallas.com/pub/PALLAS/PMB/PMB-MPI1.pdf, fetch on 2002/07/29

[Reuss00] *Reussner, R. H.:* SKaMPI: The Special Karlsruher MPI-Benchmark. User Manual, University of Karlsruhe, 2000/09/18, http://liinwww.ira.uka.de/ skampi/skampi3_userman.ps.gz, fetch on 2002/07/29

[Skill01] *Skillicorn, D.B.:* Motivating Computational Grids. http://www.cs.queensu.ca/TechReports/Reports/2001-450.ps, fetch on 2002/07/04

[Verid02] *Veridian Systems:* Portable Batch System. http://pbs.mrj.com/main.html, fetch on 2002/06/26

[Warne02] *Warnes, G. R.:* Cluster-NFS. http://clusternfs.sourceforge.net, fetch on 2002/06/24

6 1 10 19 11 1 2 1 11 11 15 15 5 7, 11 7 4 12 3 7, 28, 34, 35 13 19 18 1 11 9